PTO/SB/122 (11-08)
Approved for use through 11/30/2011. OMB 0651-0035
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

CHANGE OF CORRESPONDENCE ADDRESS
Application

Address to:
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Application Number	na
Filing Date	herewith
First Named Inventor	Steven Morein
Art Unit	na
Examiner Name	na
Attorney Docket Number	00100.36.0001

Please change the Correspondence Address for the above-identified patent application to:

☑ The address associated with Customer Number: 29153

OR

☐ Firm or Individual Name

Address

City	State	Zip

Country

Telephone	Email

This form cannot be used to change the data associated with a Customer Number. To change the data associated with an existing Customer Number use "Request for Customer Number Data Change" (PTO/SB/124).

I am the:

☐ Applicant/Inventor

☐ Assignee of record of the entire interest.
Statement under 37 CFR 3.73(b) is enclosed. (Form PTO/SB/96).

☑ Attorney or agent of record. Registration Number 34,414.

☐ Registered practitioner named in the application transmittal letter in an application without an executed oath or declaration. See 37 CFR 1.33(a)(1). Registration Number_____.

Signature	/Christopher J. Reckamp/
Typed or Printed Name	Christopher J. Reckamp
Date May 17, 2011	Telephone 312-609-7599

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below*.

☑ *Total of 1 forms are submitted.

This collection of information is required by 37 CFR 1.33. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 3 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

PTO/SB/14 (11-08)
Approved for use through 09/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	00100.36.0001
	Application Number	
Title of Invention	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER	

The application data sheet is part of the provisional or nonprovisional application for which it is being submitted. The following form contains the bibliographic data arranged in a format specified by the United States Patent and Trademark Office as outlined in 37 CFR 1.76.
This document may be completed electronically and submitted to the Office in electronic format using the Electronic Filing System (EFS) or the document may be printed and included in a paper filed application.

Secrecy Order 37 CFR 5.2

☐ Portions or all of the application associated with this Application Data Sheet may fall under a Secrecy Order pursuant to 37 CFR 5.2 (Paper filers only. Applications that fall under Secrecy Order may not be filed electronically.)

Applicant Information:

Applicant 1 Remove

Applicant Authority ◉ Inventor ○ Legal Representative under 35 U.S.C. 117 ○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Stephen	L.	Morein	

Residence Information (Select One) ◉ US Residency ○ Non US Residency ○ Active US Military Service

City	Cambridge	State/Province	MA	Country of Residence[i]	US

Citizenship under 37 CFR 1.41(b)[i] US

Mailing Address of Applicant:

Address 1	10 Magazine		
Address 2	Apt. 801		
City	Cambridge	State/Province	MA
Postal Code	02139	Country[i]	US

Applicant 2 Remove

Applicant Authority ◉ Inventor ○ Legal Representative under 35 U.S.C. 117 ○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Laurent		Lefebvre	

Residence Information (Select One) ○ US Residency ◉ Non US Residency ○ Active US Military Service

City	Lachgnaie	Country Of Residence[i]	CA

Citizenship under 37 CFR 1.41(b)[i] CA

Mailing Address of Applicant:

Address 1	124 Parenchere		
Address 2			
City	Lachgnaie	State/Province	QC
Postal Code	J6W 6A5	Country[i]	CA

Applicant 3 Remove

Applicant Authority ◉ Inventor ○ Legal Representative under 35 U.S.C. 117 ○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Andrew	E.	Gruber	

Residence Information (Select One) ◉ US Residency ○ Non US Residency ○ Active US Military Service

City	Arlington	State/Province	MA	Country of Residence[i]	US

TCL 1002

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	00100.36.0001
	Application Number	
Title of Invention	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER	

Citizenship under 37 CFR 1.41(b) i	US				
Mailing Address of Applicant:					
Address 1	215 Pleasant Street				
Address 2					
City	Arlington	**State/Province**	MA		
Postal Code	02476	**Country**i	US		

Applicant 4 Remove

Applicant Authority (•) Inventor ( ) Legal Representative under 35 U.S.C. 117 ( ) Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Andi		Skende	

Residence Information (Select One) (•) US Residency ( ) Non US Residency ( ) Active US Military Service

City	Shrewsbury	**State/Province**	MA	**Country of Residence** i	US
Citizenship under 37 CFR 1.41(b) i	US				
Mailing Address of Applicant:					
Address 1	49 Sheridan Drive, #11				
Address 2					
City	Shrewsbury	**State/Province**	MA		
Postal Code	01545	**Country**i	US		

All Inventors Must Be Listed - Additional Inventor Information blocks may be generated within this form by selecting the **Add** button. Add

Correspondence Information:

Enter either Customer Number or complete the Correspondence Information section below.
For further information see 37 CFR 1.33(a).

☐ **An Address is being provided for the correspondence Information of this application.**

Customer Number	29153		
Email Address	creckamp@vedderprice.com	Add Email	Remove Email

Application Information:

Title of the Invention	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER		
Attorney Docket Number	00100.36.0001	**Small Entity Status Claimed**	☐
Application Type	Nonprovisional		
Subject Matter	Utility		
Suggested Class (if any)		**Sub Class (if any)**	
Suggested Technology Center (if any)			
Total Number of Drawing Sheets (if any)	5	**Suggested Figure for Publication (if any)**	

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	00100.36.0001
	Application Number	
Title of Invention	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER	

Publication Information:

☐	Request Early Publication (Fee required at time of Request 37 CFR 1.219)
☐	**Request Not to Publish.** I hereby request that the attached application not be published under 35 U.S. C. 122(b) and certify that the invention disclosed in the attached application **has not and will not** be the subject of an application filed in another country, or under a multilateral international agreement, that requires publication at eighteen months after filing.

Representative Information:

Representative information should be provided for all practitioners having a power of attorney in the application. Providing this information in the Application Data Sheet does not constitute a power of attorney in the application (see 37 CFR 1.32).
Enter either Customer Number or complete the Representative Name section below. If both sections are completed the Customer Number will be used for the Representative Information during processing.

Please Select One:	◉ Customer Number	○ US Patent Practitioner	○ Limited Recognition (37 CFR 11.9)
Customer Number	29153		

Domestic Benefit/National Stage Information:

This section allows for the applicant to either claim benefit under 35 U.S.C. 119(e), 120, 121, or 365(c) or indicate National Stage entry from a PCT application. Providing this information in the application data sheet constitutes the specific reference required by 35 U.S.C. 119(e) or 120, and 37 CFR 1.78(a)(2) or CFR 1.78(a)(4), and need not otherwise be made part of the specification.

Prior Application Status	Pending		Remove
Application Number	Continuity Type	Prior Application Number	Filing Date (YYYY-MM-DD)
	Continuation of	12791597	2010-06-01

Additional Domestic Benefit/National Stage Data may be generated within this form by selecting the **Add** button. Add

Foreign Priority Information:

This section allows for the applicant to claim benefit of foreign priority and to identify any prior foreign application for which priority is not claimed. Providing this information in the application data sheet constitutes the claim for priority as required by 35 U.S.C. 119(b) and 37 CFR 1.55(a).

Remove

Application Number	Country [i]	Parent Filing Date (YYYY-MM-DD)	Priority Claimed
			○ Yes ○ No

Additional Foreign Priority Data may be generated within this form by selecting the **Add** button. Add

Assignee Information:

Providing this information in the application data sheet does not substitute for compliance with any requirement of part 3 of Title 37 of the CFR to have an assignment recorded in the Office.

Assignee 1 Remove

TCL 1002

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	00100.36.0001
	Application Number	
Title of Invention	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER	

If the Assignee is an Organization check here. ☒			
Organization Name	ATI Technologies ULC		
Mailing Address Information:			
Address 1	1 Commerce Valley Drive East		
Address 2			
City	Markham	**State/Province**	ON
Country [i] CA		Postal Code	L3T 7X6
Phone Number	905-882-2600	Fax Number	
Email Address			
Additional Assignee Data may be generated within this form by selecting the **Add** button.		Add	

Signature:

A signature of the applicant or representative is required in accordance with 37 CFR 1.33 and 10.18. Please see 37 CFR 1.4(d) for the form of the signature.

Signature	/Christopher J. Reckamp/			Date (YYYY-MM-DD)	2011-05-17
First Name	Christopher	Last Name	Reckamp	Registration Number	34414

This collection of information is required by 37 CFR 1.76. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 23 minutes to complete, including gathering, preparing, and submitting the completed application data sheet form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these records.

2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.

3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.

4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).

5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.

6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).

7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.

8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.

9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Electronic Patent Application Fee Transmittal

Application Number:	
Filing Date:	
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen L. Morein
Filer:	Christopher J. Reckamp/Christine Wright
Attorney Docket Number:	00100.36.0001

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Utility application filing	1011	1	330	330
Utility Search Fee	1111	1	540	540
Utility Examination Fee	1311	1	220	220
Pages:				
Claims:				
Independent claims in excess of 3	1201	4	220	880
Miscellaneous-Filing:				
Petition:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Miscellaneous:				
		Total in USD ($)		**1970**

Electronic Acknowledgement Receipt

EFS ID:	10111290
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen L. Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp/Christine Wright
Filer Authorized By:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001
Receipt Date:	17-MAY-2011
Filing Date:	
Time Stamp:	17:29:16
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$1970
RAM confirmation Number	4141
Deposit Account	220259
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		360001_Application.pdf	76796 5a2195ef350dfe96b37393d43d086ca74d15a3a1	yes	17

Multipart Description/PDF files in .zip description

Document Description	Start	End
Specification	1	12
Claims	13	16
Abstract	17	17

Warnings:

Information:

2	Drawings-only black and white line drawings	360001_Drawings.pdf	100418 7e6a5c9ce489409aee520309316318a7b7d231f2	no	5

Warnings:

Information:

3	Oath or Declaration filed	360001_Declaration.pdf	1711262 16d034719fb41ea904d604c9522d66d301296b4e	no	2

Warnings:

Information:

4	Change of Address	360001_Change.pdf	52028 acc3daf05193121879d529dab5b36dbed67ae54e	no	1

Warnings:

Information:

5	Application Data Sheet	360001_ADS.pdf	1032318 0457161c63792567d97d4613dade7a99db6d9934	no	5

Warnings:

Information:

6	Fee Worksheet (PTO-875)	fee-info.pdf	36605 c9542301783d22714df47e76295ef00c937337ec	no	2

Warnings:

Information:	
Total Files Size (in bytes):	3009427

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

RELATED APPLICATIONS

[0001] This application is a continuation of co-pending U.S. Application Serial No. 12/791,597, filed June 1, 2010, entitled "GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER", having as inventors Steven Morein et al., owned by instant assignee and is incorporated herein by reference, which is a continuation of co-pending U.S. Application Serial No. 11/842,256, filed August 21, 2007, entitled "GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER", having as inventors Steven Morein et al., owned by instant assignee and is incorporated herein by reference, which is a continuation of U.S. Application Serial No. 11/117,863, filed April 29, 2005, which has issued into U.S. Patent No. 7,327,369, entitled "GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER", having as inventors Steven Morein et al., and owned by instant assignee and is incorporated herein by reference which is a continuation of U.S. Application Serial No. 10/718,318, filed on November 20, 2003, which has issued into U.S. Patent No. 6,897,871, entitled "GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER", having as inventors Steven Morein et al., and owned by instant assignee and is incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The present invention generally relates to graphics processors and, more particularly, to a graphics processor architecture employing a single shader.

BACKGROUND OF THE INVENTION

[0003] In computer graphics applications, complex shapes and structures are formed through the sampling, interconnection and rendering of more simple objects, referred to as

primitives. An example of such a primitive is a triangle, or other suitable polygon. These primitives, in turn, are formed by the interconnection of individual pixels. Color and texture are then applied to the individual pixels that comprise the shape based on their location within the primitive and the primitives orientation with respect to the generated shape; thereby generating the object that is rendered to a corresponding display for subsequent viewing.

[0004] The interconnection of primitives and the application of color and textures to generated shapes are generally performed by a graphics processor. Conventional graphics processors include a series of shaders that specify how and with what corresponding attributes, a final image is drawn on a screen, or suitable display device. As illustrated in FIG. 1, a conventional shader 10 can be represented as a processing block 12 that accepts a plurality of bits of input data, such as, for example, object shape data (14) in object space (x,y,z); material properties of the object, such as color (16); texture information (18); luminance information (20); and viewing angle information (22) and provides output data (28) representing the object with texture and other appearance properties applied thereto (x´, y´, z´).

[0005] In exemplary fashion, as illustrated in FIGS. 2A-2B, the shader accepts the vertex coordinate data representing cube 30 (FIG. 2A) as inputs and provides data representing, for example, a perspectively corrected view of the cube 30´ (FIG. 2B) as an output. The corrected view may be provided, for example, by applying an appropriate transformation matrix to the data representing the initial cube 30. More specifically, the representation illustrated in FIG. 2B is provided by a vertex shader that accepts as inputs the data representing, for example, vertices V_X, V_Y and V_Z, among others of cube 30 and providing angularly oriented vertices $V_{X'}$,$V_{Y'}$ and $V_{Z'}$, including any appearance attributes of corresponding cube 30´.

[0006] In addition to the vertex shader discussed above, a shader processing block that operates on the pixel level, referred to as a pixel shader is also used when generating an object for display. Generally, the pixel shader provides the color value associated with each pixel of a rendered object. Conventionally, both the vertex shader and pixel shader are separate components that are configured to perform only a single transformation or operation. Thus, in order to perform a position and a texture transformation of an input, at least two shading operations and hence, at least two shaders, need to be employed. Conventional graphics processors require the use of both a vertex shader and a pixel shader in order to generate an object. Because both types of shaders are required, known graphics processors are relatively large in size, with most of the real estate being taken up by the vertex and pixel shaders.

[0007] In addition to the real estate penalty associated with conventional graphics processors, there is also a corresponding performance penalty associated therewith. In conventional graphics processors, the vertex shader and the pixel shader are juxtaposed in a sequential, pipelined fashion, with the vertex shader being positioned before and operating on vertex data before the pixel shader can operate on individual pixel data.

[0008] Thus, there is a need for an improved graphics processor employing a shader that is both space efficient and computationally effective.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention and the associated advantages and features thereof, will become better understood and appreciated upon review of the following detailed description of the invention, taken in conjunction with the following drawings, where like numerals represent like elements, in which:

[0010] FIG. 1 is a schematic block diagram of a conventional shader;

[0011] FIGS. 2A-2B are graphical representations of the operations performed by the shader illustrated in FIG. 1;

[0012] FIG. 3 is a schematic block diagram of a conventional graphics processor architecture;

[0013] FIG. 4A is a schematic block diagram of a graphics processor architecture according to the present invention;

[0014] FIG. 4B is a schematic block diagram of an optional input component to the graphics processor according to an alternate embodiment of the present invention; and

[0015] FIG. 5 is an exploded schematic block diagram of the unified shader employed in the graphics processor illustrated in FIG. 4A.

DETAILED DESCRIPTION OF THE INVENTION

[0016] Briefly stated, the present invention is directed to a graphics processor that employs a unified shader that is capable of performing both the vertex operations and the pixel operations in a space saving and computationally efficient manner. In an exemplary embodiment, a graphics processor according to the present invention includes an arbiter circuit for selecting one of a plurality of inputs for processing in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs.

[0017] The shader includes a general purpose register block for storing at least the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations and a processor capable of executing

both floating point arithmetic and logical operations on the selected inputs according to the instructions maintained in the sequencer. The shader of the present invention is referred to as a "unified" shader because it is configured to perform both vertex and pixel operations. By employing the unified shader of the present invention, the associated graphics processor is more space efficient than conventional graphics processors because the unified shader takes up less real estate than the conventional multi-shader processor architecture.

[0018] In addition, according to the present invention, the unified shader is more computationally efficient because it allows the shader to be flexibly allocated to pixels or vertices based on workload.

[0019] Referring now to FIG. 3, illustrated therein is a graphics processor incorporating a conventional pipeline architecture. As shown, the graphics processor 40 includes a vertex fetch block 42 which receives vertex information relating to a primitive to be rendered from an off-chip memory 55 on line 41. The fetched vertex data is then transmitted to a vertex cache 44 for storage on line 43. Upon request, the vertex data maintained in the vertex cache 44 is transmitted to a vertex shader 46 on line 45. As discussed above, an example of the information that is requested by and transmitted to the vertex shader 46 includes the object shape, material properties (e.g. color), texture information, and viewing angle. Generally, the vertex shader 46 is a programmable mechanism which applies a transformation position matrix to the input position information (obtained from the vertex cache 44), thereby providing data representing a perspectively corrected image of the object to be rendered, along with any texture or color coordinates thereof.

[0020] After performing the transformation operation, the data representing the transformed vertices are then provided to a vertex store 48 on line 47. The vertex store 48 then

transmits the modified vertex information contained therein to a primitive assembly block 50 on line 49. The primitive assembly block 50 assembles, or converts, the input vertex information into a plurality of primitives to be subsequently processed. Suitable methods of assembling the input vertex information into primitives is known in the art and will not be discussed in greater detail here. The assembled primitives are then transmitted to a rasterization engine 52, which converts the previously assembled primitives into pixel data through a process referred to as walking. The resulting pixel data is then transmitted to a pixel shader 54 on line 53.

[0021] The pixel shader 54 generates the color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels. In addition, the pixel shader 54 is capable of fetching texture data from a texture map 57 as indexed by the pixel data from the rasterization engine 52 by transmitting such information on line 55 to the texture map. The requested texture data is then transmitted back from the texture map 57 on line 57´ and stored in a texture cache 56 before being routed to the pixel shader on line 58. Once the texture data has been received, the pixel shader 54 then performs specified logical or arithmetic operations on the received texture data to generate the pixel color or other appearance attribute of interest. The generated pixel appearance attribute is then combined with a base color, as provided by the rasterization engine on line 53, to thereby provide a pixel color to the pixel corresponding at the position of interest. The pixel appearance attribute present on line 59 is then transmitted to post raster processing blocks (not shown).

[0022] As described above, the conventional graphics processor 40 requires the use of two separate shaders: a vertex shader 46 and a pixel shader 54. A drawback associated with such an architecture is that the overall footprint of the graphics processor is relatively large as the two

shaders take up a large amount of real estate. Another drawback associated with conventional graphics processor architectures is that can exhibit poor computational efficiency.

[0023] Referring now to FIG. 4A, in an exemplary embodiment, the graphics processor 60 of the present invention includes a multiplexer 66 having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine 74 provided at a second input. A control signal generated by an arbiter 64 is transmitted to the multiplexer 66 on line 63. The arbiter 64 determines which of the two inputs to the multiplexer 66 is transmitted to a unified shader 62 for further processing. The arbitration scheme employed by the arbiter 64 is as follows: the vertex data on the first input of the multiplexer 66 is transmitted to the unified shader 62 on line 65 if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader 62 for further processing.

[0024] Referring briefly to FIG. 5, the unified shader 62 will now be described. As illustrated, the unified shader 62 includes a general purpose register block 92, a plurality of source registers: including source register A 93, source register B 95, and source register C 97, a processor (e.g. CPU) 96 and a sequencer 99. The general purpose register block 92 includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer 66 on line 65 or any other information to be maintained within the unified shader. The data present in the general purpose register block 92 is transmitted to the plurality of source registers via line 109.

[0025] The processor 96 may be comprised of a dedicated piece of hardware or can be configured as part of a general purpose computing device (i.e. personal computer). In an

exemplary embodiment, the processor 96 is adapted to perform 32-bit floating point arithmetic operations as well as a complete series of logical operations on corresponding operands. As shown, the processor is logically partitioned into two sections. Section 96 is configured to execute, for example, the 32-bit floating point arithmetic operations of the unified shader. The second section, 96A, is configured to perform scaler operations (e.g. log, exponent, reciprocal square root) of the unified shader.

[0026] The sequencer 99 includes constants block 91 and an instruction store 98. The constants block 91 contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store 98 contains the necessary instructions that are executed by the processor 96 in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block 92 as provided by the source registers 93-95. The instruction store 98 further includes memory fetch instructions that, when executed, causes the unified shader 62 to fetch texture and other types of data, from memory 82 (FIG. 4A). In operation, the sequencer 99 determines whether the next instruction to be executed (from the instruction store 98) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer 99 sends the request to a fetch block (not shown) which retrieves the required information from memory 82 (FIG. 4A). The retrieved information is then transmitted to the sequencer 99, through the vertex texture cache 68 (FIG. 4A) as described in greater detail below.

[0027] If the next instruction to be executed is an arithmetic or logical instruction, the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block 92 into the appropriate source registers (93, 95, 97) for execution, and an appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are

to be executed on the several operands present in the source registers. At this point, the processor 96 executes the instructions on the operands present in the source registers and provides the result on line 85. The information present on line 85 may be transmitted back to the general purpose register block 92 for storage, or transmitted to succeeding components of the graphics processor 60.

[0028] As discussed above, the instruction store 98 maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader 99 of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader 62 of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.

[0029] The unified shader 62 has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store 98, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block 92 from multiplexer 66. The instruction store 98 then passes the corresponding control signals to the processor 96 on line 101 to perform such vertex operations. However, if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor 96 are continued, based on the instructions

maintained in the instruction store 98, until enough registers within the general purpose register block 92 become available. Thus, through the sharing of resources within the unified shader 62, processing of image data is enhanced as there is no down time associated with the processor 96.

[0030] Referring back to FIG. 4A, the graphics processor 60 further includes a cache block 70, including a parameter cache 70A and a position cache 70B which accepts the pixel based output of the unified shader 62 on line 85 and stores the respective pixel parameter and position information in the corresponding cache. The pixel information present in the cache block 70 is then transmitted to the primitive assembly block 72 on line 71. The primitive assembly block 72 is responsible for assembling the information transmitted thereto from the cache block 70 into a series of triangles, or other suitable primitives, for further processing. The assembled primitives are then transmitted on line 73 to rasterization engine block 74, where the transmitted primitives are then converted into individual pixel data information through a walking process, or any other suitable pixel generation process. The resulting pixel data from the rasterization engine block 74 is the interpolated pixel parameter data that is transmitted to the second input of the multiplexer 66 on line 75.

[0031] In those situations when vertex data is transmitted to the unified shader 62 through the multiplexer 66, the resulting vertex data generated by the processor 96, is transmitted to a render back end block 76 which converts the resulting vertex data into at least one of several formats suitable for later display on display device 84. For example, if a stained glass appearance effect is to be applied to an image, the information corresponding to such appearance effect is associated with the appropriate position data by the render back end 76. The information from the render back end 76 is then transmitted to memory 82 and a display

controller line 80 via memory controller 78. Such appropriately formatted information is then transmitted on line 83 for presentation on display device 84.

[0032] Referring now to FIG. 4B, shown therein is a vertex block 61 which is used to provide the vertex information at the first input of the multiplexer 66 according to an alternate embodiment of the present invention. The vertex block 61 includes a vertex fetch block 61A which is responsible for retrieving vertex information from memory 82, if requested, and transmitting that vertex information into the vertex cache 61B. The information stored in the vertex cache 61B comprises the vertex information that is coupled to the first input of multiplexer 66.

[0033] As discussed above, the graphics processor 60 of the present invention incorporates a unified shader 62 which is capable of performing both vertex manipulation operations and pixel manipulation operations based on the instructions stored in the instruction store 98. In this fashion, the graphics processor 60 of the present invention takes up less real estate than conventional graphics processors as separate vertex shaders and pixel shaders are no longer required. In addition, as the unified shader 62 is capable of alternating between performing vertex manipulation operations and pixel manipulation operations, graphics processing efficiency is enhanced as one type of data operations is not dependent upon another type of data operations. Therefore, any performance penalties experienced as a result of dependent operations in conventional graphics processors are overcome.

[0034] The above detailed description of the present invention and the examples described therein have been presented for the purposes of illustration and description. It is therefore contemplated that the present invention cover any and all modifications, variations and

equivalents that fall within the spirit and scope of the basic underlying principles disclosed and claimed herein.

CLAIMS

What is claimed is:

1. A method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

2. A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.

3. A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

4. A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

5. A unified shader comprising:

a processor unit;

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

6. The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.

7. The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.

8. The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs.

9. The shader of claim 5, wherein the processor unit executes vertex calculations while the pixel calculations are still in progress.

10. The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.

11. The shader of claim 7, wherein the control signal is provided by an arbiter.

12. A graphics processor comprising:

a unified shader comprising a processor unit that executes vertex calculations while the pixel calculations are still in progress.

13. The graphics processor of claim 12 wherein the unified shader comprises a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to

cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

14. The graphics processor of claim 12 comprising a vertex block operative to fetch vertex information from memory.

15. A unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.

16. The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

[0035] A graphics processing architecture employing a single shader is disclosed. The architecture includes a circuit operative to select one of a plurality of inputs in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs. The shader includes a register block which is used to store the plurality of selected inputs, a sequencer which maintains vertex manipulation and pixel manipulations instructions and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs in response to the instructions maintained in the sequencer.

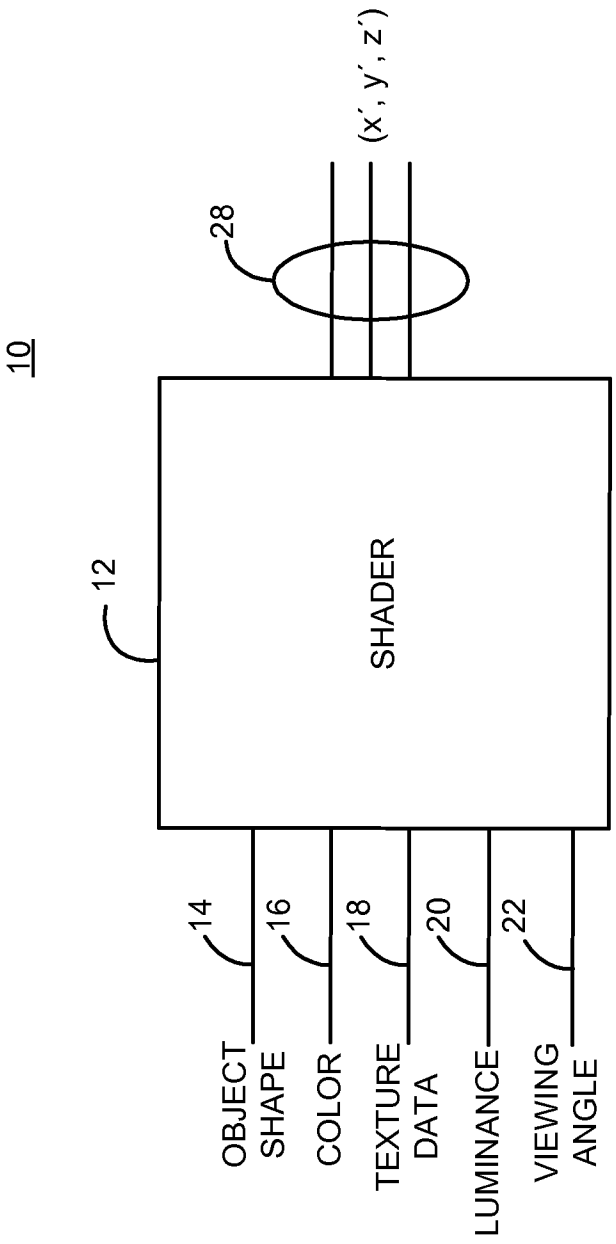


FIG. 1
(PRIOR ART)

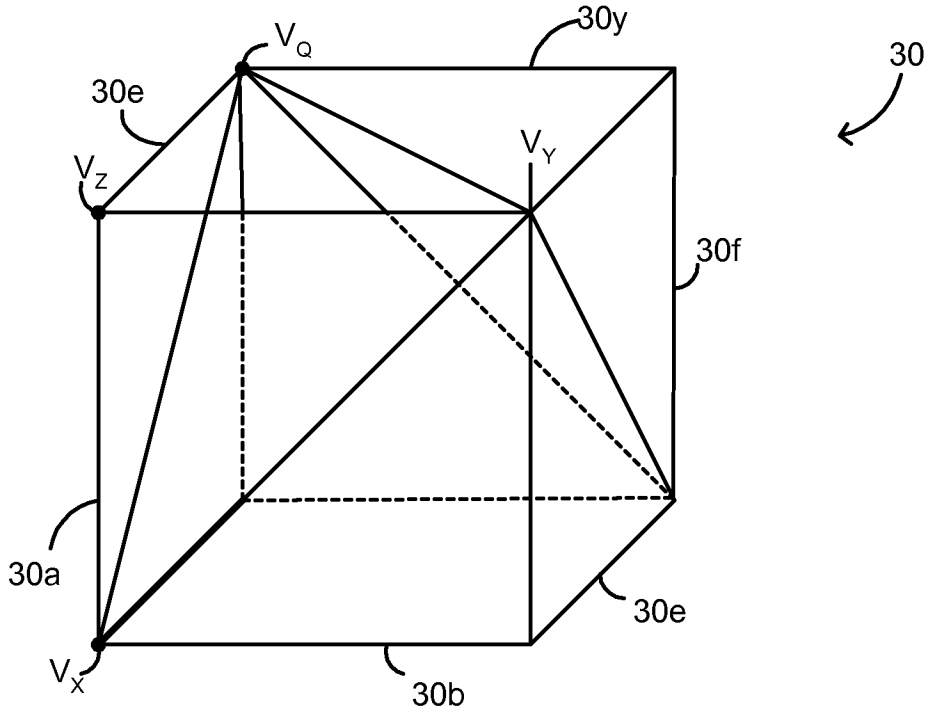


FIG. 2A
(PRIOR ART)

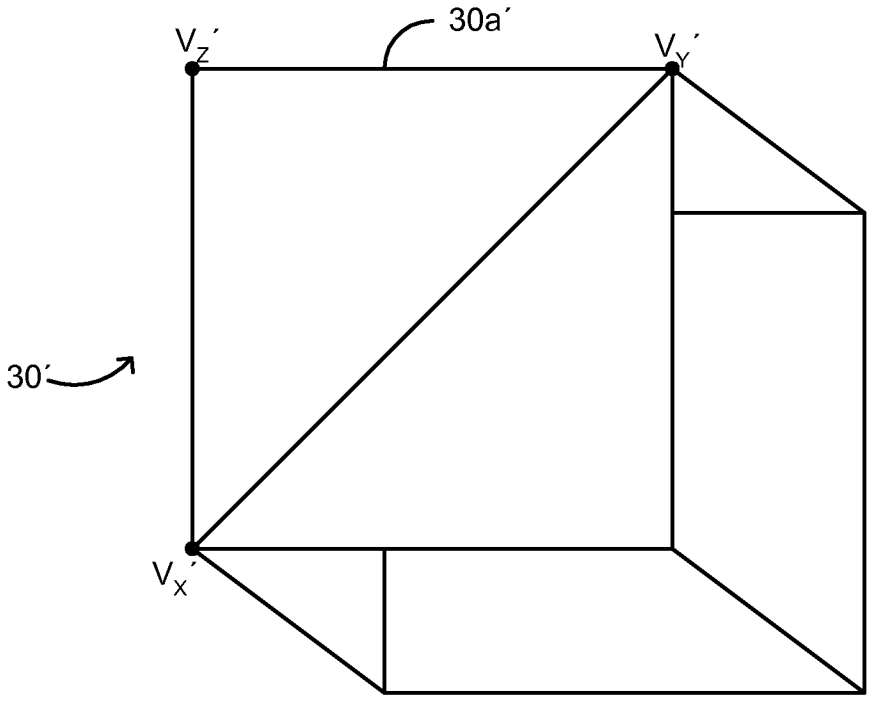


FIG. 2B
(PRIOR ART)

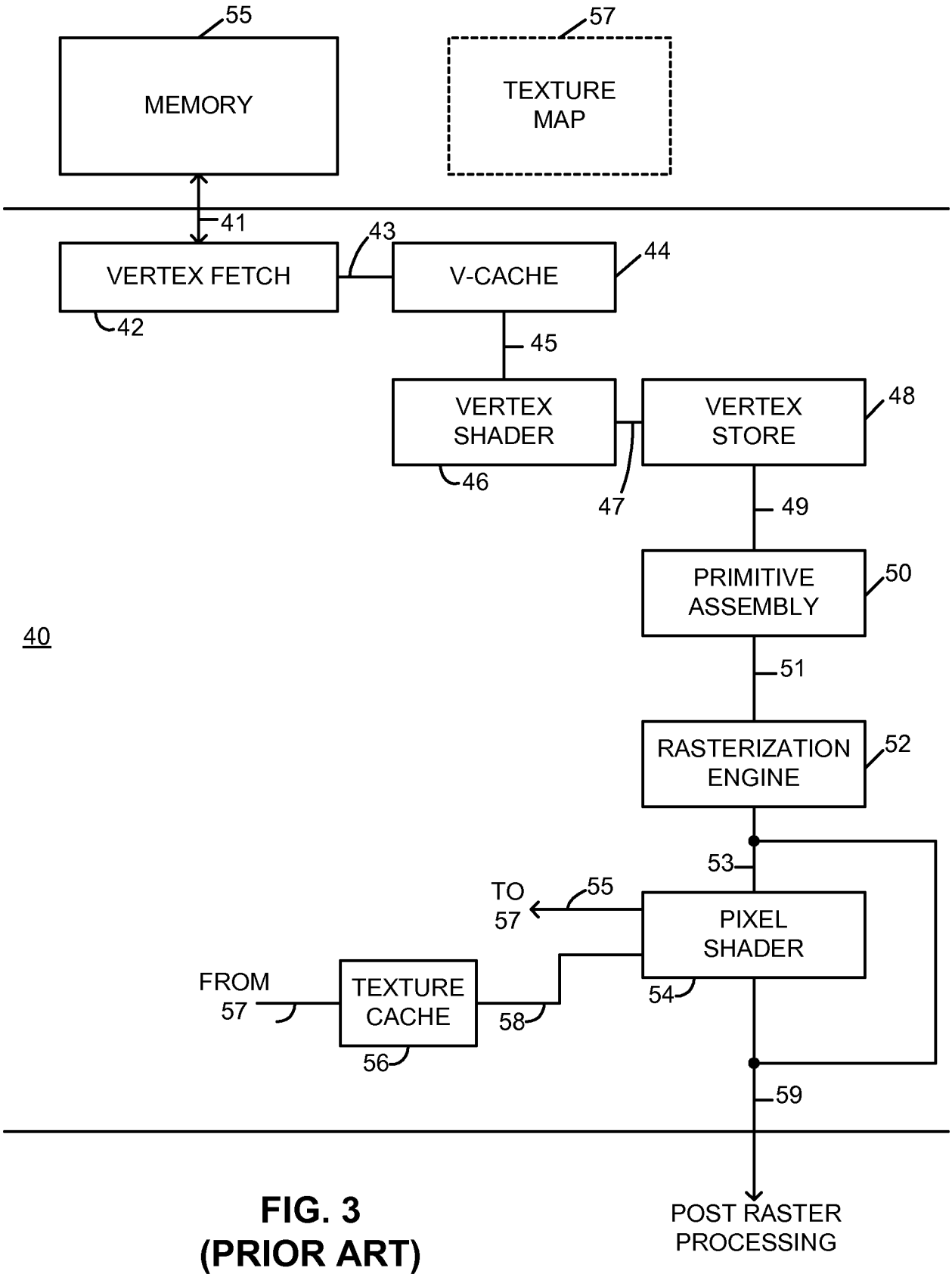


FIG. 3
(PRIOR ART)

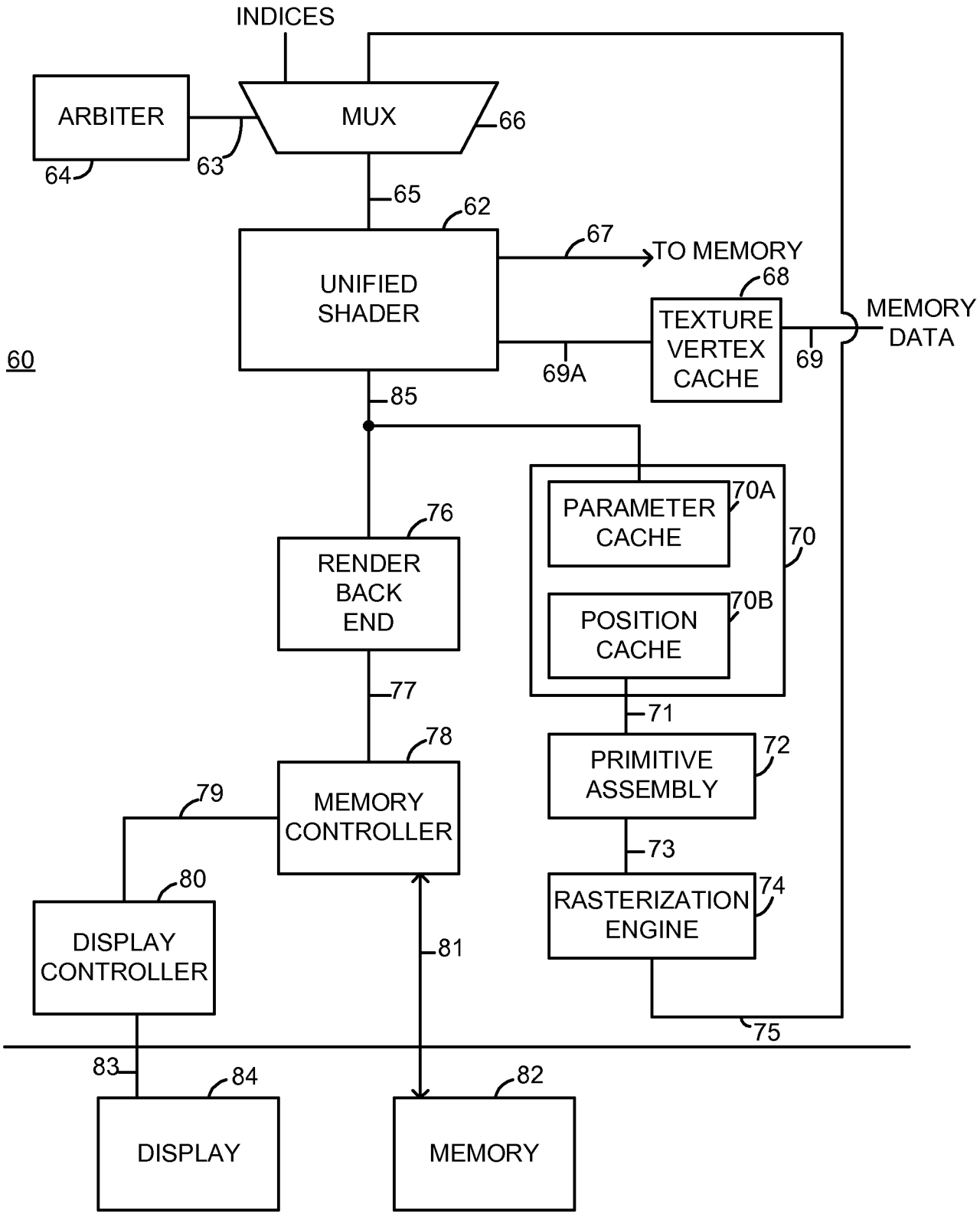

INDICES
ARBITER
MUX
66
64
63
65
62
UNIFIED SHADER
67
TO MEMORY
68
TEXTURE VERTEX CACHE
MEMORY DATA
69
69A
60
85
PARAMETER CACHE
70A
70
POSITION CACHE
70B
76
RENDER BACK END
77
71
78
PRIMITIVE ASSEMBLY
72
79
MEMORY CONTROLLER
73
80
RASTERIZATION ENGINE
74
DISPLAY CONTROLLER
81
75
83
84
82
DISPLAY
MEMORY

FIG. 4A

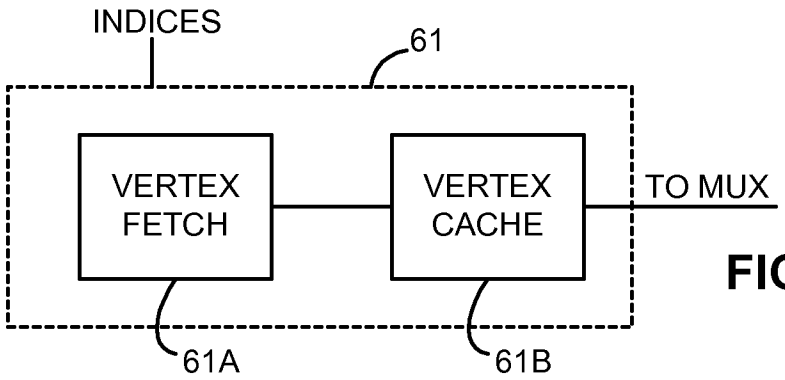

INDICES
61
VERTEX FETCH
VERTEX CACHE
TO MUX
61A
61B

FIG. 4B

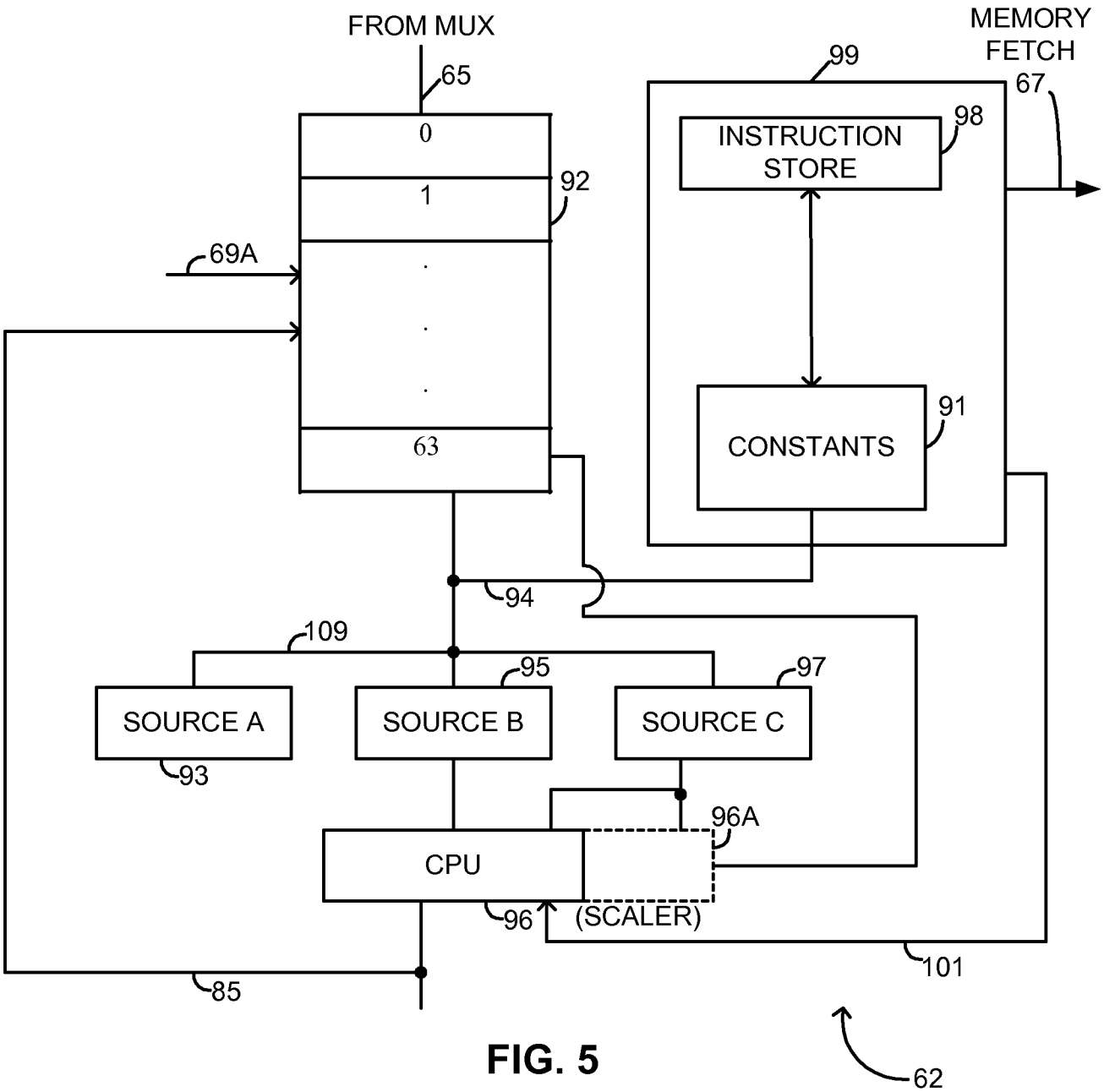

FROM MUX
65
0
1
63
92
69A
MEMORY FETCH
67
99
INSTRUCTION STORE
98
CONSTANTS
91
94
109
95
97
SOURCE A
SOURCE B
SOURCE C
93
96A
CPU
96
(SCALER)
101
85
62

FIG. 5

DECLARATION
FOR UTILITY OR DESIGN
PATENT APPLICATION
(37 CFR 1.63)

☒ Declaration Submitted with Initial Filing, OR
☐ Declaration Submitted after Initial Filing (surcharge (37 CFR 1.16(e)) required)

Attorney Docket Number 0100.02.0001
First Named Inventor: Morein et al.
COMPLETE IF KNOWN
Application Number: Unknown
Filing Date: _____
Group Art Unit: Unknown
Examiner Name: Unknown

As a below named inventor, I hereby declare that:
My residence, post office address, and citizenship are as stated below next to my name.
I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: **A GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

the specification of which:
☒ ...is attached hereto.
☐ ...was filed on _____ as United States Application Number _____ or as PCT International Application Number _____ and was amended on (MM/DD/YYYY) _____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.
I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Copy Attached? YES	NO
			☐	☐	☐
			☐	☐	☐

☐ Additional foreign application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

I hereby claim the benefit under 35 U.S.C. 119(e) of any United States provisional application(s) listed below.

Application Number(s)	Filing Data (MM/DD/YYYY)

☐ Additional provisional application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

U.S. Parent Application or PCT Parent Number	Parent Filing Date (MM/DD/YYYY)	Parent Patent Number (if applicable)

☐ Additional U.S. or PCT international application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

As a named inventor, I hereby appoint the following registered practitioner(s) to prosecute this application and to transaction all business in the Patent and Trademark Office connected therewith:

Name	Registration Number	Name	Registration Number
Christopher J. Reckamp	34,414	Angelo J. Bufalino	29,622
Joseph P. Krause	32,578	Robert Beiser	28,687
Michael J. Turgeon	39,404	Brent A. Boyd	51,020
Timothy J. Bechen	48,126		

☐ Additional registered practitioner(s) named on supplemental Registered Practitioner Information sheet PTO/SB/02C attached hereto.\

Direct all correspondence to:

Vedder, Price, Kaufman & Kammholz
222 N. LaSalle Street, Suite 2600
Chicago, Illinois 60601
Telephone: 312-609-7500
Facsimile: 312-609-5005

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Name of Sole or First Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])		Family Name or Surname	
Steven		Morein	
Inventor's Signature		**Date**	
Residence	City: Cambridge State: ~~Maine~~ MA	Country: US	Citizenship: US
Post Office Address	10 Magazine, Apt. 801		
City: Cambridge	State: ~~Maine~~ MA	ZIP: 02139	Country: US

Name of Additional Joint Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])		Family Name or Surname	
Laurent		Lefebvre	
Inventor's Signature		**Date**	8/21/03
Residence	City: Lachenaie State: Qc	Country: CANADA	Citizenship: CANADA
Post Office Address	124 PARENCHERE		
City:	State: Qc	ZIP J6W 6A5	Country: CANADA

Name of Additional Joint Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])		Family Name or Surname	
Andy		Gruber	
Inventor's Signature		**Date** 8/20/03	
Residence	City: Arlington State: ~~Maine~~ MA	Country: US	Citizenship: US
Post Office Address	215 Pleasant Street		
City: Arlington	State: ~~Maine~~ MA	ZIP 02476	Country: US

Name of Additional Joint Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])		Family Name or Surname	
Andi		Skende	
Inventor's Signature		**Date** 08/20/2003	
Residence	City: Shrewsbury State: ~~Maine~~ MA	Country: US	Citizenship: US
Post Office Address	49 Sheridan Drive, #11		
City: Shrewsbury	State: ~~Maine~~ MA	ZIP 01545	Country: US

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number: 13/109,738

APPLICATION AS FILED - PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY RATE($)	SMALL ENTITY FEE($)	OR	OTHER THAN SMALL ENTITY RATE($)	OTHER THAN SMALL ENTITY FEE($)
BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	330
SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	540
EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	220
TOTAL CLAIMS (37 CFR 1.16(i))	16 minus 20 =	*			OR	x 52 =	0.00
INDEPENDENT CLAIMS (37 CFR 1.16(h))	7 minus 3 =	* 4				x 220 =	880
APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $270 ($135 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						0.00
MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							0.00
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	1970

APPLICATION AS AMENDED - PART II

		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE($)	SMALL ENTITY ADDITIONAL FEE($)	OR	OTHER THAN SMALL ENTITY RATE($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE($)
AMENDMENT A	Total (37 CFR 1.16(i))	*	Minus	**	=	x =		OR	x =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	x =		OR	x =	
	Application Size Fee (37 CFR 1.16(s))									
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE($)	ADDITIONAL FEE($)		RATE($)	ADDITIONAL FEE($)
AMENDMENT B	Total (37 CFR 1.16(i))	*	Minus	**	=	x =		OR	x =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	x =		OR	x =	
	Application Size Fee (37 CFR 1.16(s))									
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest found in the appropriate box in column 1.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING or 371(c) DATE	GRP ART UNIT	FIL FEE REC'D	ATTY.DOCKET.NO	TOT CLAIMS	IND CLAIMS
13/109,738	05/17/2011	2628	1970	00100.36.0001	16	7

CONFIRMATION NO. 2020

29153
ADVANCED MICRO DEVICES, INC.
C/O VEDDER PRICE P.C.
222 N.LASALLE STREET
CHICAGO, IL 60601

FILING RECEIPT

OC000000047943624

Date Mailed: 06/01/2011

Receipt is acknowledged of this non-provisional patent application. The application will be taken up for examination in due course. Applicant will be notified as to the results of the examination. Any correspondence concerning the application must include the following identification information: the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please submit a written request for a Filing Receipt Correction. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections**

Applicant(s)

Stephen Morein, Cambridge, MA;
Laurent Lefebvre, Lachgnaie, CANADA;
Andy Gruber, Arlington, MA;
Andi Skende, Shrewsbury, MA;

Assignment For Published Patent Application

ATI TECHNOLOGIES ULC, Markham, CANADA

Power of Attorney:

Robert Beiser--28687
Angelo Bufalino--29622
Joseph Krause--32578
Christopher Reckamp--34414
Michael Turgeon--39404
Timothy Bechen--48126
Brent Boyd--51020

Domestic Priority data as claimed by applicant

This application is a CON of 12/791,597 06/01/2010
which is a CON of 11/842,256 08/21/2007 ABN
which is a CON of 11/117,863 04/29/2005 PAT 7,327,369
which is a CON of 10/718,318 11/20/2003 PAT 6,897,871

Foreign Applications (You may be eligible to benefit from the **Patent Prosecution Highway** program at the USPTO. Please see http://www.uspto.gov for more information.)

If Required, Foreign Filing License Granted: 05/27/2011

The country code and number of your priority application, to be used for filing abroad under the Paris Convention, is **US 13/109,738**

Projected Publication Date: 09/08/2011

Non-Publication Request: No

Early Publication Request: No

Title

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

Preliminary Class

345

PROTECTING YOUR INVENTION OUTSIDE THE UNITED STATES

Since the rights granted by a U.S. patent extend only throughout the territory of the United States and have no effect in a foreign country, an inventor who wishes patent protection in another country must apply for a patent in a specific country or in regional patent offices. Applicants may wish to consider the filing of an international application under the Patent Cooperation Treaty (PCT). An international (PCT) application generally has the same effect as a regular national patent application in each PCT-member country. The PCT process **simplifies** the filing of patent applications on the same invention in member countries, but **does not result** in a grant of "an international patent" and does not eliminate the need of applicants to file additional documents and fees in countries where patent protection is desired.

Almost every country has its own patent law, and a person desiring a patent in a particular country must make an application for patent in that country in accordance with its particular laws. Since the laws of many countries differ in various respects from the patent law of the United States, applicants are advised to seek guidance from specific foreign countries to ensure that patent rights are not lost prematurely.

Applicants also are advised that in the case of inventions made in the United States, the Director of the USPTO must issue a license before applicants can apply for a patent in a foreign country. The filing of a U.S. patent application serves as a request for a foreign filing license. The application's filing receipt contains further information and guidance as to the status of applicant's license for foreign filing.

Applicants may wish to consult the USPTO booklet, "General Information Concerning Patents" (specifically, the section entitled "Treaties and Foreign Patents") for more information on timeframes and deadlines for filing foreign patent applications. The guide is available either by contacting the USPTO Contact Center at 800-786-9199, or it can be viewed on the USPTO website at http://www.uspto.gov/web/offices/pac/doc/general/index.html.

For information on preventing theft of your intellectual property (patents, trademarks and copyrights), you may wish to consult the U.S. Government website, http://www.stopfakes.gov. Part of a Department of Commerce initiative, this website includes self-help "toolkits" giving innovators guidance on how to protect intellectual property in specific countries such as China, Korea and Mexico. For questions regarding patent enforcement issues, applicants may call the U.S. Government hotline at 1-866-999-HALT (1-866-999-4158).

LICENSE FOR FOREIGN FILING UNDER

Title 35, United States Code, Section 184

Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Bureau of Industry and Security, Department of Commerce (15 CFR parts 730-774); the Office of Foreign AssetsControl, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

PTO/SB/08a (01-10)
Approved for use through 07/31/2012. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Not for submission under 37 CFR 1.99)

Application Number	13109738
Filing Date	2011-05-17
First Named Inventor	Stephen Morein
Art Unit	2628
Examiner Name	na
Attorney Docket Number	00100.36.0001

U.S.PATENTS Remove

Examiner Initial*	Cite No	Patent Number	Kind Code[1]	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	5550962		1996-08-27	Nakamura et al.	
	2	5818469		1998-10-06	Lawless et al.	
	3	6118452		2000-09-12	Gannett	
	4	6353439		2002-03-05	Lindholm et al.	
	5	6384824		2002-05-07	Morgan et al.	
	6	6417858		2002-07-09	Bosch et al.	
	7	6573893		2003-06-03	Naqvi et al.	
	8	6650327		2002-11-18	Airey et al.	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)		
	Application Number	13109738
	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

	9	6650330		2003-11-18	Lindholm et al.	
	10	6704018		2004-03-09	Mori et al.	
	11	6724394		2004-04-20	Zatz et al.	
	12	6731289		2004-05-04	Peercy et al.	
	13	6809732		2004-10-26	Zatz et al.	
	14	6864893		2005-03-08	Zatz	
	15	6897871		2005-05-24	Morein et al.	
	16	6980209		2005-12-27	Donham et al.	
	17	7015913		2006-03-21	Lindholm et al.	
	18	7038685		2006-05-02	Lindholm	
	19	7327369		2008-02-05	Morein et al.	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)		
	Application Number	13109738
	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

	20	5485559		1996-01-16	Sakaibara et al.	
	21	7239322	B2	2007-07-03	Lefebvre et al.	
	22	7746348	B2	2010-06-29	Lefebvre et al.	
	23	7742053	B2	2010-06-22	Lefebvre et al.	

If you wish to add additional U.S. Patent citation information please click the Add button. Add

U.S.PATENT APPLICATION PUBLICATIONS Remove

Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	20030076320	A1	2003-04-24	Collodi	
	2	20030164830	A1	2003-09-04	Kent	
	3	20040041814	A1	2004-03-04	Wyatt et al.	
	4	20040164987	A1	2004-08-26	Aronson et al.	
	5	20050068325	A1	2005-03-31	Lefebvre et al.	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)		
	Application Number	13109738
	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

	6	20100231592	A1	2010-09-16	Morein et al.	
	7	20030030643	A1	2003-02-13	Taylor et al.	
	8	20070222785	A1	2007-09-27	Lefebvre et al.	
	9	20070222787	A1	2007-09-27	Lefebvre et al.	
	10	20050200629	A1	2005-09-15	Morein et al.	
	11	20070222786	A1	2007-09-27	Lefebvre et al.	
	12	20070285427	A1	2007-12-13	Morein et al.	
	13	20100156915	A1	2010-06-24	Lefebvre et al.	

If you wish to add additional U.S. Published Application citation information please click the Add button. Add

FOREIGN PATENT DOCUMENTS Remove

Examiner Initial*	Cite No	Foreign Document Number[3]	Country Code[2] i	Kind Code[4]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear	T[5]
	1	2299408	EP	A2	2011-03-23	Morein et al.		☐

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)		
	Application Number	13109738
	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

	2	2309460	EP	A1	2011-04-13	Morein et al.		☐
	3	2296116	EP	A2	2011-03-16	Morein et al.		☐

If you wish to add additional Foreign Patent Document citation information please click the Add button Add

NON-PATENT LITERATURE DOCUMENTS Remove

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T[5]
	1	European Patent Office Examination Report; EP Application No. 04798938.9; dated November 9, 2006; pages 1-3.	☐
	2	PURCELL, TIMOTHY J. et al.; Ray Tracing on Programmable Graphics Hardware; SIGGRAPH '02; San Antonio, TX; ACM Transactions on Graphics; July 2002; vol. 21, no. 3; pgs. 703-712.	☐
	3	MARK, WILLIAM R. et al.; CG: A system for programming graphics hardware in a C-like language; SIGGRAPH '03; San Diego, CA; ACM Transactions on Graphics; July 2002; vol. 22, no. 3; pgs. 896-907.	☐
	4	BRETERNITZ, JR., MAURICIO et al.; Compilation, Architectural Support, and Evaluation of SIMD Graphics Pipeline Programs on a General-Purpose CPU; IEEE; 2003; pgs. 1-11.	☐
	5	International Search Report and Written Opinion; International Application No. PCT/IB2004/003821; dated March 22, 2005.	☐
	6	EP Supplemental Search Report; EP Application No. 10075688.1; dated February 25, 2011.	☐
	7	EP Supplemental Search Report; EP Application No. 10075686.5; dated February 25, 2011.	☐

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)		
	Application Number	13109738
	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

	8	EP Supplemental Search Report; EP Application No. 10075687.3; dated February 25, 2011.	☐
	9	EP Supplemental Search Report; EP Application No. 10075685.7; dated February 25, 2011.	☐
	10	ELDRIDGE, MATTHEW et al.; Pomegranate: A Fully Scalable Graphics Architecture; Computer Graphics, SIGGRAPH 2000 Conference Proceedings; July 23, 2000.	☐
	11	OWENS, JOHN D. et al.; Polygon Rendering on a Stream Architecture; Proceedings 2000 SIGGRAPH/Eurographics Workshop on Graphics Hardware; August 21, 2000.	☐
	12	Chinese Office Action; Chinese Application No. 2004800405708; dated September, 2008.	☐
	13	Chinese Office Action; Chinese Application No. 2004800405708; dated November, 2009.	☐
	14	Chinese Office Action; Chinese Application No. 2004800405708; dated September, 2010	☐

If you wish to add additional non-patent literature document citation information please click the Add button Add

EXAMINER SIGNATURE

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. [2] Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). [3] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [4] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [5] Applicant is to place a check mark here if English language translation is attached.

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)		
	Application Number	13109738
	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

CERTIFICATION STATEMENT

Please see 37 CFR 1.97 and 1.98 to make the appropriate selection(s):

☐ That each item of information contained in the information disclosure statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of the information disclosure statement. See 37 CFR 1.97(e)(1).

OR

☐ That no item of information contained in the information disclosure statement was cited in a communication from a foreign patent office in a counterpart foreign application, and, to the knowledge of the person signing the certification after making reasonable inquiry, no item of information contained in the information disclosure statement was known to any individual designated in 37 CFR 1.56(c) more than three months prior to the filing of the information disclosure statement. See 37 CFR 1.97(e)(2).

☒ See attached certification statement.

☐ Fee set forth in 37 CFR 1.17 (p) has been submitted herewith.

☐ None

SIGNATURE

A signature of the applicant or representative is required in accordance with CFR 1.33, 10.18. Please see CFR 1.4(d) for the form of the signature.

Signature	/Christopher J. Reckamp/	Date (YYYY-MM-DD)	2011-07-14
Name/Print	Christopher J. Reckamp	Registration Number	34,414

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 1 hour to complete, including gathering, preparing and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these record s.

2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.

3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.

4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).

5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.

6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).

7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.

8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.

9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 2 299 408 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
23.03.2011 Bulletin 2011/12

(51) Int Cl.:
G06T 15/00 *(2011.01)* ***G06T 15/80*** *(2011.01)*

(21) Application number: **10075687.3**

(22) Date of filing: **19.11.2004**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IS IT LI LU MC NL PL PT RO SE SI SK TR
Designated Extension States:
AL HR LT MK YU

(30) Priority: **20.11.2003 US 718318**

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
04798938.9 / 1 706 847

(71) Applicant: **ATI Technologies Inc.**
Markham,
Ontario L3T 7X6 (CA)

(72) Inventors:
- **Morein, Steven**
  Cambridge, MA 02139 (US)
- **Lefebvre, Laurent**
  Lachenaie
  Quebec J6W 6A5 (CA)
- **Gruber, Andy**
  Arlington, MA 02476 (US)
- **Skende, Andi**
  Shrewsbury, MA 01545 (US)

(74) Representative: **Waldren, Robin Michael**
Marks & Clerk LLP
90 Long Acre
London
WC2E 9RA (GB)

Remarks:
This application was filed on 01-10-2010 as a divisional application to the application mentioned under INID code 62.

(54) A graphics processing architecture employing a unified shader

(57) A graphics processor, comprising: an arbiter circuit for selecting one of a plurality of inputs in response to a control signal; and a shader, coupled to the arbiter circuit, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and performing one of the vertex operations or pixel operations based on the selected one of the plurality of inputs, wherein the shader provides a appearance attribute.

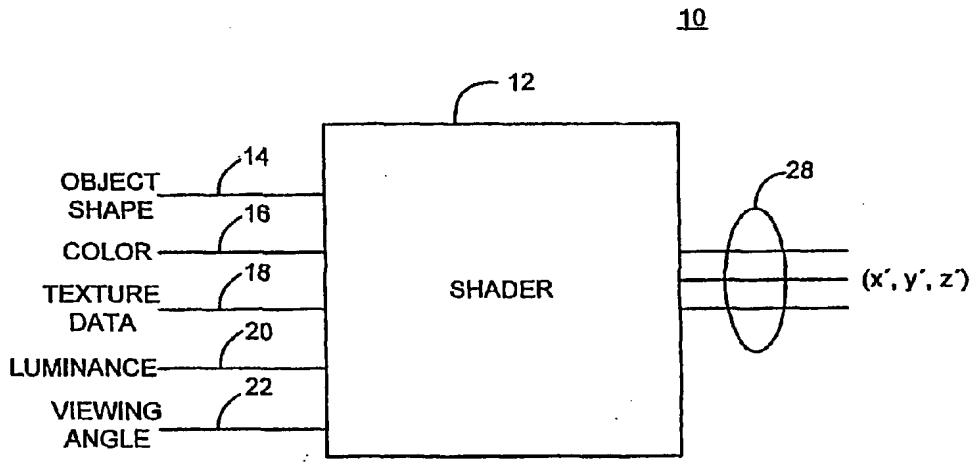


FIG. 1
(PRIOR ART)

Printed by Jouve, 75001 PARIS (FR)

TCL 1002

Description

FIELD OF THE INVENTION

[0001] The present invention generally relates to graphics processors and, more particularly, to a graphics processor architecture employing a single shader.

BACKGROUND OF THE INVENTION

[0002] In computer graphics applications, complex shapes and structures are formed through the sampling, interconnection and rendering of more simple objects, referred to as primitives. An example of such a primitive is a triangle, or other suitable polygon. These primitives, in turn, are formed by the interconnection of individual pixels. Color and texture are then applied to the individual pixels that comprise the shape based on their location within the primitive and the primitives orientation with respect to the generated shape; thereby generating the object that is rendered to a corresponding display for subsequent viewing.

[0003] The interconnection of primitives and the application of color and textures to generated shapes are generally performed by a graphics processor. Conventional graphics processors include a series of shaders that specify how and with what corresponding attributes, a final image is drawn on a screen, or suitable display device. As illustrated in FIG. 1, a conventional shader 10 can be represented as a processing block 12 that accepts a plurality of bits of input data, such as, for example, object shape data (14) in object space (x,y,z); material properties of the object, such as color (16); texture information (18); luminance information (20); and viewing angle information (22) and provides output data (28) representing the object with texture and other appearance properties applied thereto (x', y', z').

[0004] In exemplary fashion, as illustrated in FIGS. 2A-2B, the shader accepts the vertex coordinate data representing cube 30 (FIG. 2A) as inputs and provides data representing, for example, a perspectively corrected view of the cube 30' (FIG. 2B) as an output. The corrected view may be provided, for example, by applying an appropriate transformation matrix to the data representing the initial cube 30. More specifically, the representation illustrated in FIG. 2B is provided by a vertex shader that accepts as inputs the data representing, for example, vertices V_x, V_y and V_z, among others of cube 30 and providing angularly oriented vertices $V_{x'}$,$V_{y'}$ and $V_{z'}$, including any appearance attributes of corresponding cube 30'.

[0005] In addition to the vertex shader discussed above, a shader processing block that operates on the pixel level, referred to as a pixel shader is also used when generating an object for display. Generally, the pixel shader provides the color value associated with each pixel of a rendered object. Conventionally, both the vertex shader and pixel shader are separate components that are configured to perform only a single transformation or operation. Thus, in order to perform a position and a texture transformation of an input, at least two shading operations and hence, at least two shaders, need to be employed. Conventional graphics processors require the use of both a vertex shader and a pixel shader in order to generate an object. Because both types of shaders are required, known graphics processors are relatively large in size, with most of the real estate being taken up by the vertex and pixel shaders.

[0006] In addition to the real estate penalty associated with conventional graphics processors, there is also a corresponding performance penalty associated therewith. In conventional graphics processors, the vertex shader and the pixel shader are juxtaposed in a sequential, pipelined fashion, with the vertex shader being positioned before and operating on vertex data before the pixel shader can operate on individual pixel data.

[0007] Thus, there is a need for an improved graphics processor employing a shader that is both space efficient and computationally effective.

SUMMARY OF THE INVENTION

[0008] Briefly stated, the present invention is directed to a graphics processor that employs a unified shader that is capable of performing both the vertex operations and the pixel operations in a space saving and computationally efficient manner. In an exemplary embodiment, a graphics processor according to the present invention includes an arbiter circuit for selecting one of a plurality of inputs for processing in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs.

[0009] The shader includes a general purpose register block for storing at least the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs according to the instructions maintained in the sequencer. The shader of the present invention is referred to as a "unified" shader because it is configured to perform both vertex and pixel operations. By employing the unified shader of the present invention, the associated graphics processor is more space efficient than conventional graphics processors because the unified shader takes up less real estate than the conventional multi-shader processor architecture.

[0010] In addition, according to the present invention, the unified shader is more computationally efficient because it allows the shader to be flexibly allocated to pixels or vertices based on workload.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention and the associated advantages and features thereof, will become better understood and appreciated upon review of the following detailed description of the invention, taken in conjunction with the following drawings, where like numerals represent like elements, in which:

FIG. 1 is a schematic block diagram of a conventional shader;

FIGS. 2A-2B are graphical representations of the operations performed by the shader illustrated in FIG. 1;

FIG. 3 is a schematic block diagram of a conventional graphics processor architecture;

FIG. 4A is a schematic block diagram of a graphics processor architecture according to the present invention;

FIG. 4B is a schematic block diagram of an optional input component to the graphics processor according to an alternate embodiment of the present invention; and

FIG. 5 is an exploded schematic block diagram of the unified shader employed in the graphics processor illustrated in FIG. 4A.

DETAILED DESCRIPTION OF THE INVENTION

[0012] FIG. 3, illustrates a graphics processor incorporating a conventional pipeline architecture. As shown, the graphics processor 40 includes a vertex fetch block 42 which receives vertex information relating to a primitive to be rendered from an off-chip memory 55 on line 41. The fetched vertex data is then transmitted to a vertex cache 44 for storage on line 43. Upon request, the vertex data maintained in the vertex cache 44 is transmitted to a vertex shader 46 on line 45. As discussed above, an example of the information that is requested by and transmitted to the vertex shader 46 includes the object shape, material properties (e.g. color), texture information, and viewing angle. Generally, the vertex shader 46 is a programmable mechanism which applies a transformation position matrix to the input position information (obtained from the vertex cache 44), thereby providing data representing a perspectively corrected image of the object to be rendered, along with any texture or color coordinates thereof.

[0013] After performing the transformation operation, the data representing the transformed vertices are then provided to a vertex store 48 on line 47. The vertex store 48 then transmits the modified vertex information contained therein to a primitive assembly block 50 on line 49. The primitive assembly block 50 assembles, or converts, the input vertex information into a plurality of primitives to be subsequently processed. Suitable methods of assembling the input vertex information into primitives is known in the art and will not be discussed in greater detail here. The assembled primitives are then transmitted to a rasterization engine 52, which converts the previously assembled primitives into pixel data through a process referred to as walking. The resulting pixel data is then transmitted to a pixel shader 54 on line 53.

[0014] The pixel shader 54 generates the color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels. In addition, the pixel shader 54 is capable of fetching texture data from a texture map 57 as indexed by the pixel data from the rasterization engine 52 by transmitting such information on line 55 to the texture map. The requested texture data is then transmitted back from the texture map 57 on line 57' and stored in a texture cache 56 before being routed to the pixel shader on line 58. Once the texture data has been received, the pixel shader 54 then performs specified logical or arithmetic operations on the received texture data to generate the pixel color or other appearance attribute of interest. The generated pixel appearance attribute is then combined with a base color, as provided by the rasterization engine on line 53, to thereby provide a pixel color to the pixel corresponding at the position of interest. The pixel appearance attribute present on line 59 is then transmitted to post raster processing blocks (not shown).

[0015] As described above, the conventional graphics processor 40 requires the use of two separate shaders: a vertex shader 46 and a pixel shader 54. A drawback associated with such an architecture is that the overall footprint of the graphics processor is relatively large as the two shaders take up a large amount of real estate. Another drawback associated with conventional graphics processor architectures is that can exhibit poor computational efficiency.

[0016] Referring now to FIG. 4A, in an exemplary embodiment, the graphics processor 60 of the present invention includes a multiplexer 66 having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine 74 provided at a second input. A control signal generated by an arbiter 64 is transmitted to the multiplexer 66 on line 63. The arbiter 64 determines which of the two inputs to the multiplexer 66 is transmitted to a unified shader 62 for further processing. The arbitration scheme employed by the arbiter 64 is as follows: the vertex data on the first input of the multiplexer 66 is transmitted to the unified shader 62 on line 65 if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader 62 for further processing.

[0017] Referring briefly to FIG. 5, the unified shader

62 will now be described. As illustrated, the unified shader 62 includes a general purpose register block 92, a plurality of source registers: including source register A 93, source register B 95, and source register C 97, a processor (e.g. CPU) 96 and a sequencer 99. The general purpose register block 92 includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer 66 on line 65 or any other information to be maintained within the unified shader. The data present in the general purpose register block 92 is transmitted to the plurality of source registers via line 109.

[0018] The processor 96 may be comprised of a dedicated piece of hardware or can be configured as part of a general purpose computing device (i.e, personal computer). In an exemplary embodiment, the processor 96 is adapted to perform 32-bit floating point arithmetic operations as well as a complete series of logical operations on corresponding operands. As shown, the processor is logically partitioned into two sections. Section 96 is configured to execute, for example, the 32-bit floating point arithmetic operations of the unified shader. The second section, 96A, is configured to perform scaler operations (e.g. log, exponent, reciprocal square root) of the unified shader.

[0019] The sequencer 99 includes constants block 91 and an instruction store 98. The constants block 91 contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store 98 contains the necessary instructions that are executed by the processor 96 in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block 92 as provided by the source registers 93-95. The instruction store 98 further includes memory fetch instructions that, when executed, causes the unified shader 62 to fetch texture and other types of data, from memory 82 (FIG. 4A). In operation, the sequencer 99 determines whether the next instruction to be executed (from the instruction store 98) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer 99 sends the request to a fetch block (not shown) which retrieves the required information from memory 82 (FIG. 4A). The retrieved information is then transmitted to the sequencer 99, through the vertex texture cache 68 (FIG. 4A) as described in greater detail below.

[0020] If the next instruction to be executed is an arithmetic or logical instruction, the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block 92 into the appropriate source registers (93, 95, 97) for execution, and an appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor 96 executes the instructions on the operands present in the source registers and provides the result on line 85. The information present on line 85 may be transmitted back to the general purpose register block 92 for storage, or transmitted to succeeding components of the graphics processor 60.

[0021] As discussed above, the instruction store 98 maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader 99 of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader 62 of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.

[0022] The unified shader 62 has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store 98, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block 92 from multiplexer 66. The instruction store 98 then passes the corresponding control signals to the processor 96 on line 101 to perform such vertex operations. However, if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor 96 are continued, based on the instructions maintained in the instruction store 98, until enough registers within the general purpose register block 92 become available. Thus, through the sharing of resources within the unified shader 62, processing of image data is enhanced as there is no down time associated with the processor 96.

[0023] Referring back to FIG. 4A, the graphics processor 60 further includes a cache block 70, including a parameter cache 70A and a position cache 70B which accepts the pixel based output of the unified shader 62 on line 85 and stores the respective pixel parameter and position information in the corresponding cache. The pixel information present in the cache block 70 is then transmitted to the primitive assembly block 72 on line 71. The primitive assembly block 72 is responsible for assembling the information transmitted thereto from the cache block 70 into a series of triangles, or other suitable primitives, for further processing. The assembled primitives are then transmitted on line 73 to rasterization engine block 74, where the transmitted primitives are then converted into individual pixel data information through a walking process, or any other suitable pixel generation process. The resulting pixel data from the rasterization engine block 74 is the interpolated pixel parameter data that is transmitted to the second input of the multiplexer 66 on line 75.

[0024] In those situations when vertex data is transmitted to the unified shader 62 through the multiplexer 66, the resulting vertex data generated by the processor 96, is transmitted to a render back end block 76 which converts the resulting vertex data into at least one of several formats suitable for later display on display device 84. For example, if a stained glass appearance effect is to be applied to an image, the information corresponding to such appearance effect is associated with the appropriate position data by the render back end 76. The information from the render back end 76 is then transmitted to memory 82 and a display controller line 80 via memory controller 78. Such appropriately formatted information is then transmitted on line 83 for presentation on display device 84.

[0025] Referring now to FIG. 4B, shown therein is a vertex block 61 which is used to provide the vertex information at the first input of the multiplexer 66 according to an alternate embodiment of the present invention. The vertex block 61 includes a vertex fetch block 61A which is responsible for retrieving vertex information from memory 82, if requested, and transmitting that vertex information into the vertex cache 61 B. The information stored in the vertex cache 61 B comprises the vertex information that is coupled to the first input of multiplexer 66.

[0026] As discussed above, the graphics processor 60 of the present invention incorporates a unified shader 62 which is capable of performing both vertex manipulation operations and pixel manipulation operations based on the instructions stored in the instruction store 98. In this fashion, the graphics processor 60 of the present invention takes up less real estate than conventional graphics processors as separate vertex shaders and pixel shaders are no longer required. In addition, as the unified shader 62 is capable of alternating between performing vertex manipulation operations and pixel manipulation operations, graphics processing efficiency is enhanced as one type of data operations is not dependent upon another type of data operations. Therefore, any performance penalties experienced as a result of dependent operations in conventional graphics processors are overcome.

[0027] The above detailed description of the present invention and the examples described therein have been presented for the purposes of illustration and description. It is therefore contemplated that the present invention cover any and all modifications, variations and equivalents that fall within the scope of the basic underlying principles disclosed and claimed herein.

Claims

1. A graphics processor, comprising: an arbiter circuit for selecting one of a plurality of inputs in response to a control signal; and a shader, coupled to the arbiter circuit, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and performing one of the vertex operations or pixel operations based on the selected one of the plurality of inputs, wherein the shader provides a appearance attribute.

2. The graphics processor of claim 1, further including a vertex storage block for maintaining vertex information.

3. The graphics processor of claim 2, wherein the vertex storage block further includes a parameter cache operative to maintain appearance attribute data for a corresponding vertex and a position cache operative to maintain position data for a corresponding vertex.

4. The graphics processor of claim 1, wherein the appearance attribute is color, and the color is associated with a corresponding pixel when the selected one of the plurality inputs is pixel data.

5. The graphics processor of claim 1, wherein the appearance attribute is position, and the position attribute is associated with a corresponding vertex when the selected one of the plurality of inputs is vertex data.

6. The graphics processor of claim 5, wherein the appearance attribute is color, and the color attribute is associated with a corresponding pixel when the selected one of the plurality of inputs is pixel data.

7. The graphics processor of claim 5, wherein the appearance attribute is one of the following: color, lighting, texture, normal and position data.

8. The graphics processor of claim 1, wherein the appearance value is depth.

9. The graphics processor of claim 1, further including a selection circuit, wherein the selection circuit is a multiplexer, and the control signal is provided by an arbiter, wherein the arbiter is coupled to the multiplexer.

10. The graphics processor of claim 1, wherein the shader provides vertex position data and further including a primitive assembly block, coupled to the shader, operative to generate primitives in response to the vertex position data.

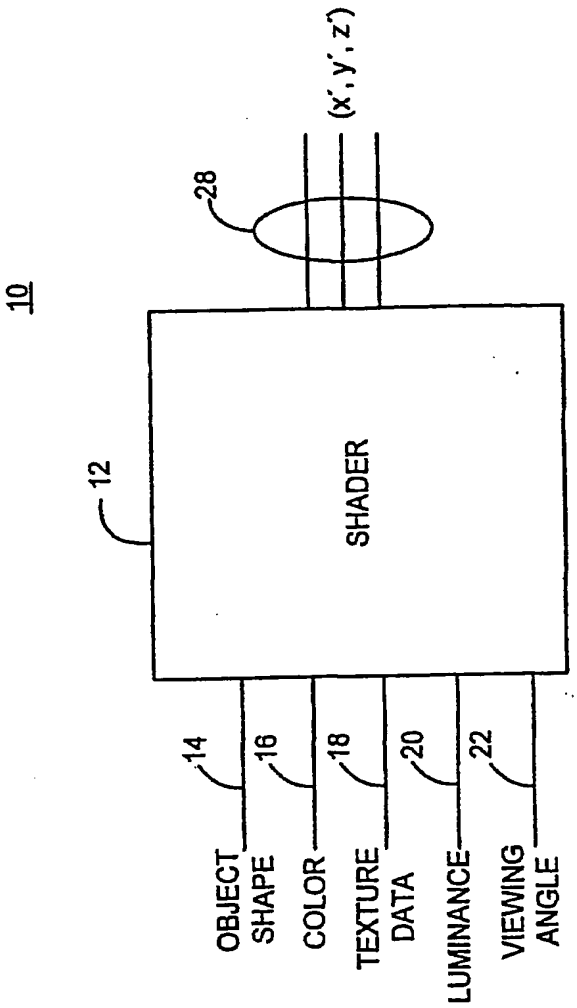


FIG. 1
(PRIOR ART)

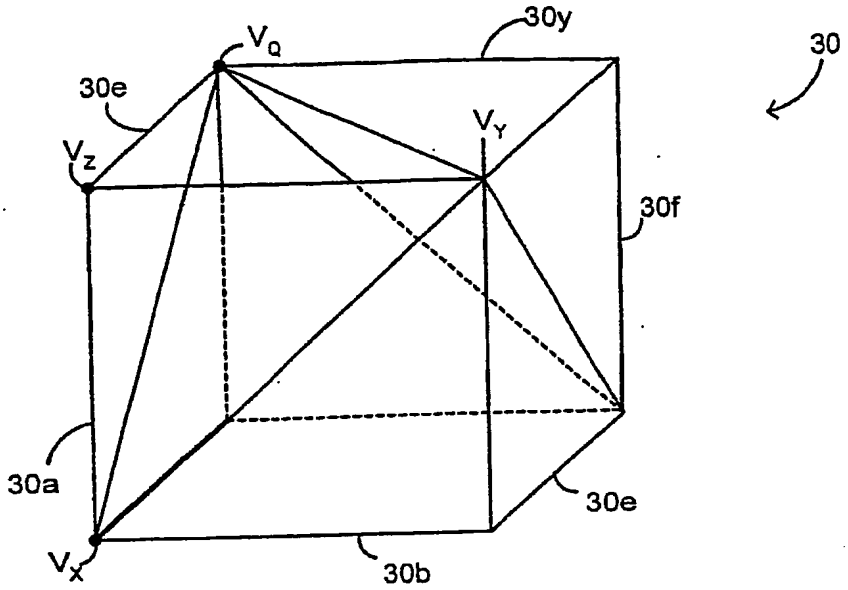


FIG. 2A
(PRIOR ART)

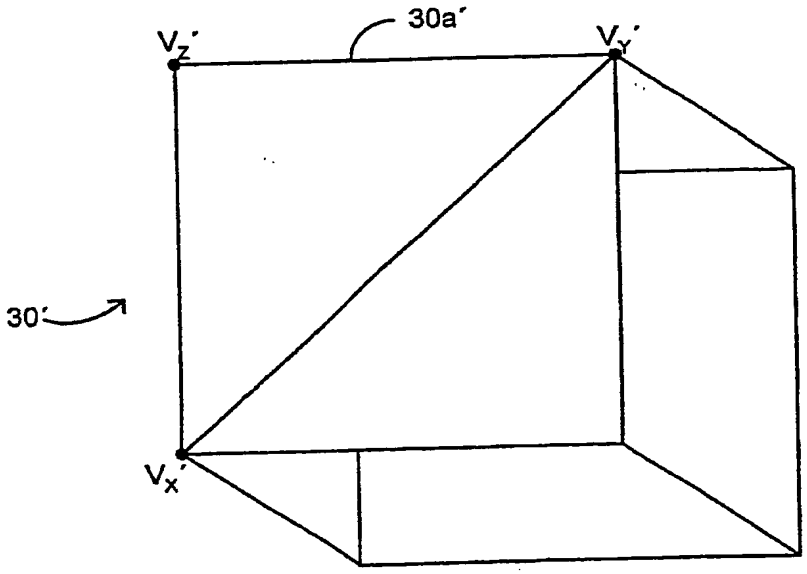


FIG. 2B
(PRIOR ART)

TCL 1002

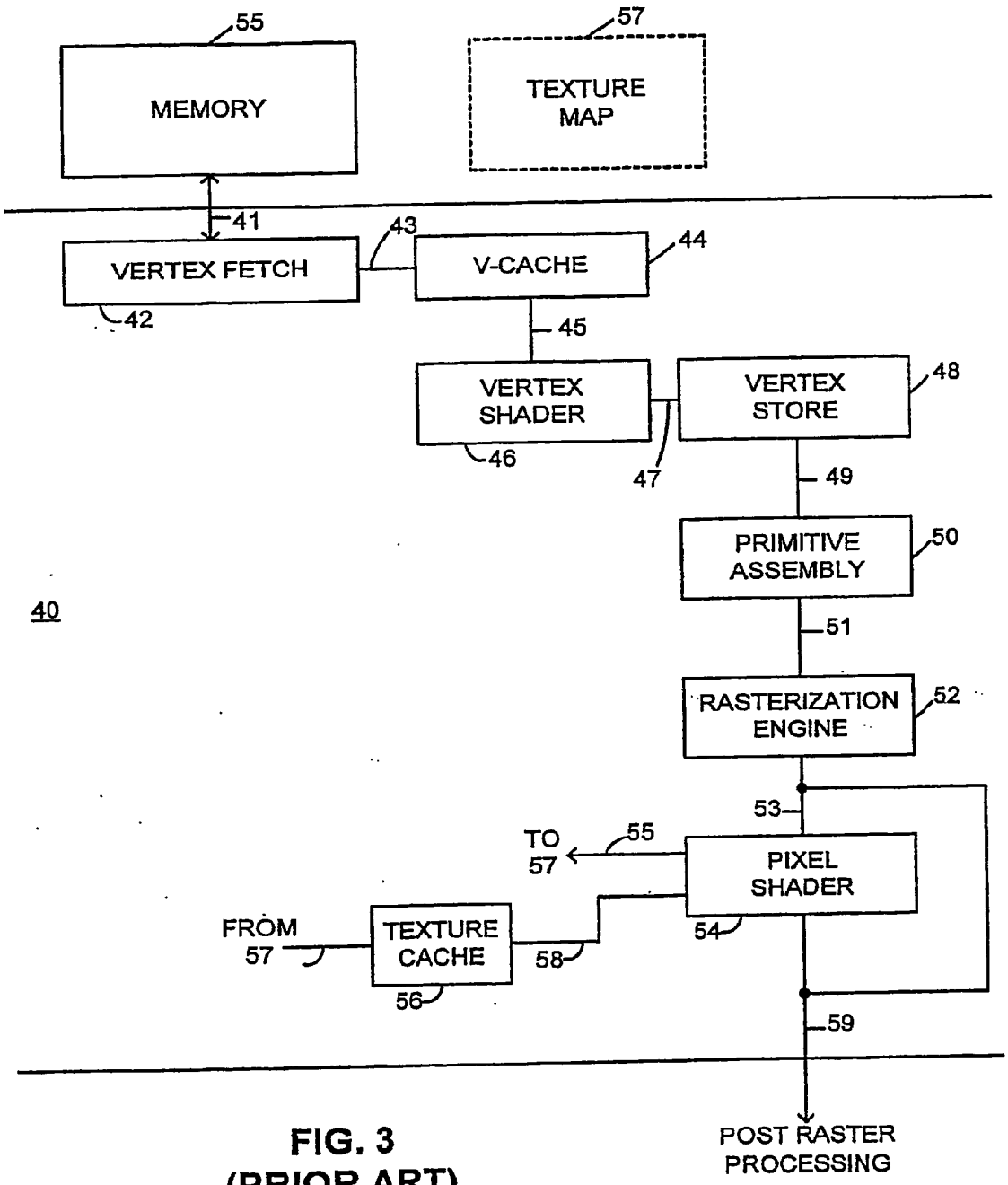


FIG. 3
(PRIOR ART)

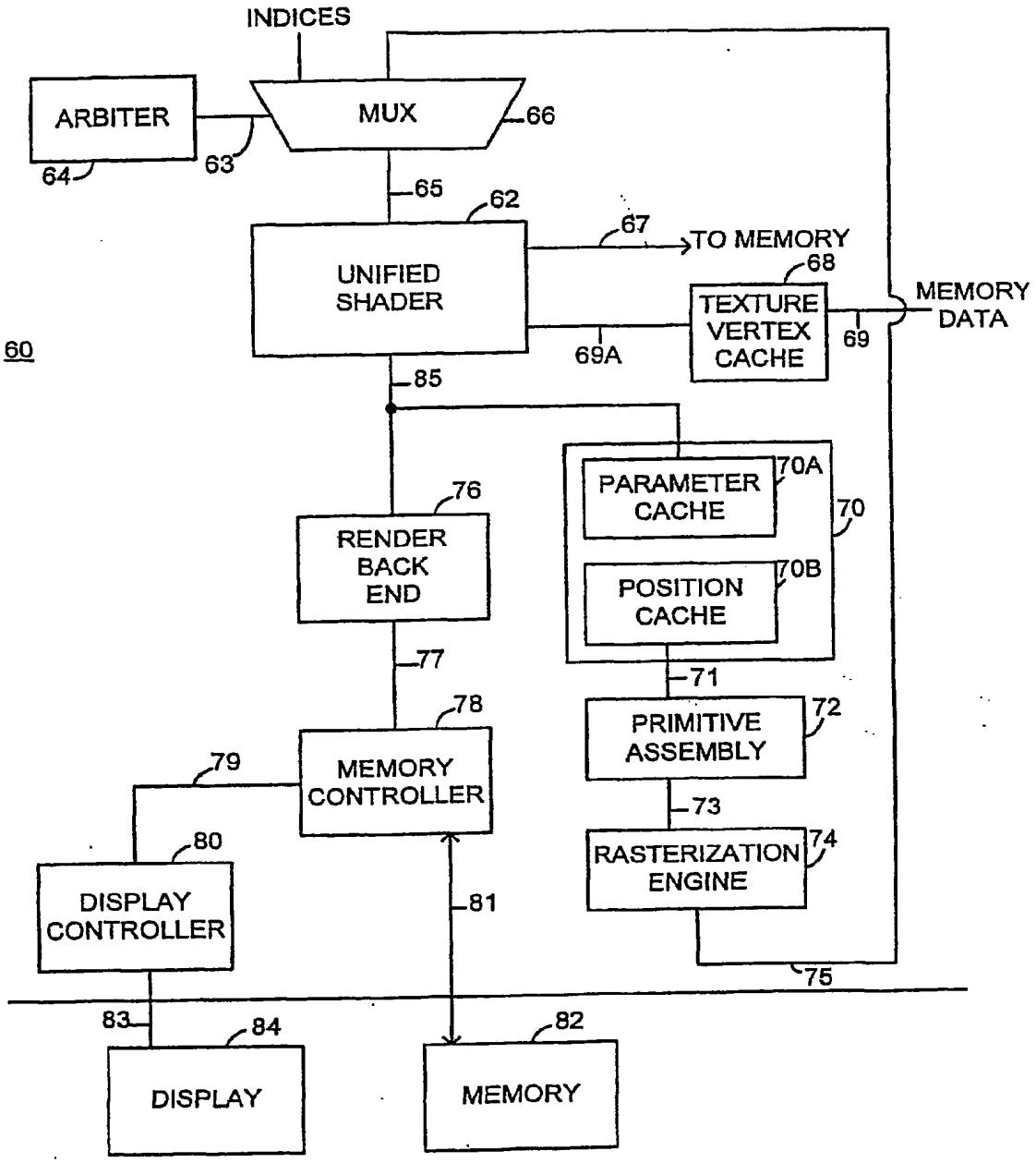

INDICES
ARBITER
MUX
66
64
63
65
62
67
TO MEMORY
UNIFIED SHADER
68
TEXTURE VERTEX CACHE
MEMORY DATA
69
69A
60
85
PARAMETER CACHE
70A
70
POSITION CACHE
70B
76
RENDER BACK END
77
71
78
PRIMITIVE ASSEMBLY
72
79
MEMORY CONTROLLER
73
80
RASTERIZATION ENGINE
74
DISPLAY CONTROLLER
81
75
83
84
82
DISPLAY
MEMORY

FIG. 4A

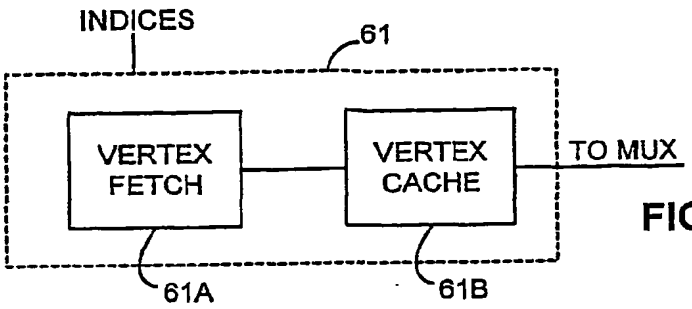

INDICES
61
VERTEX FETCH
VERTEX CACHE
TO MUX
61A
61B

FIG. 4B

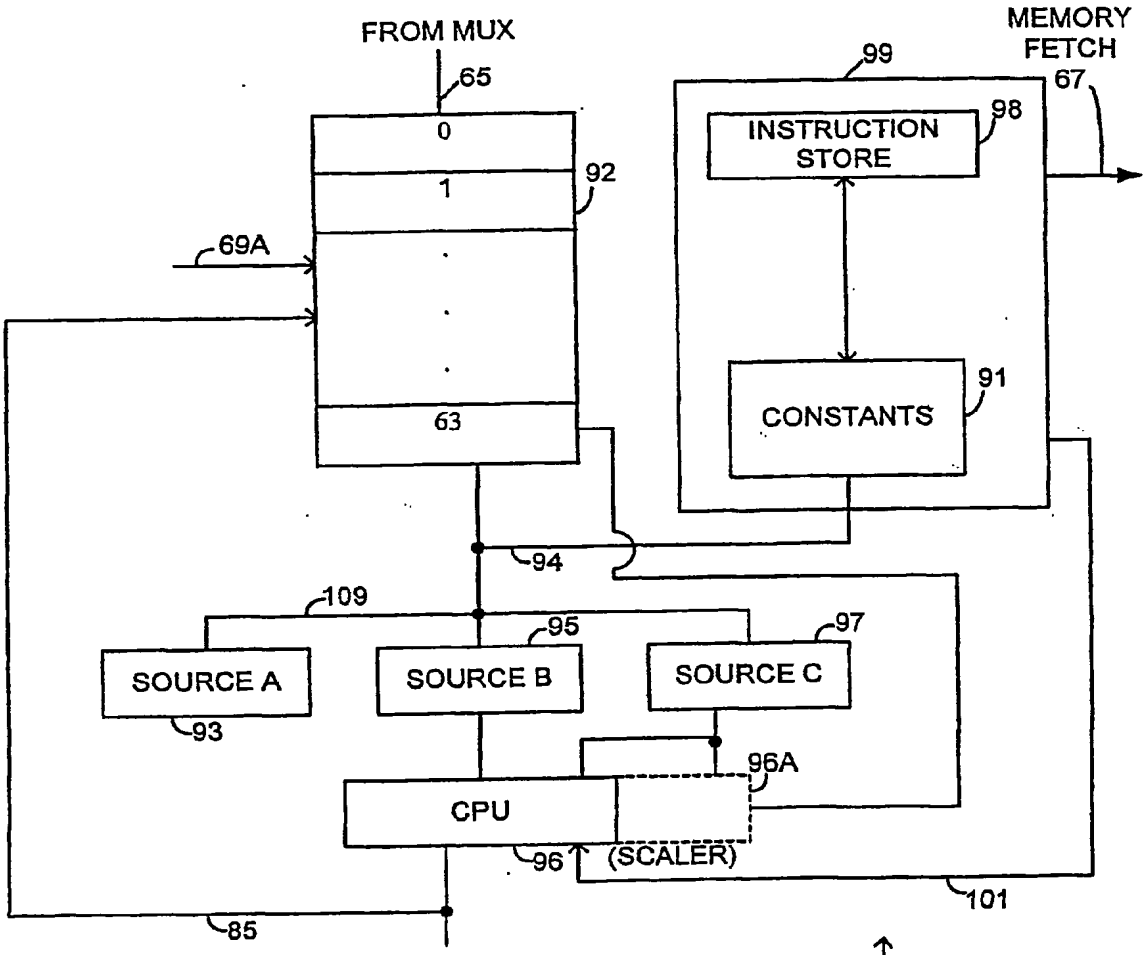

FROM MUX
65
0
1
63
92
69A
MEMORY FETCH
67
99
INSTRUCTION STORE
98
CONSTANTS
91
94
109
95
97
SOURCE A
93
SOURCE B
SOURCE C
96A
CPU
96
(SCALER)
101
85
62

FIG. 5

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 2 309 460 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
13.04.2011 Bulletin 2011/15

(51) Int Cl.:
G06T 15/00 *(2011.01)* ***G06T 15/80*** *(2011.01)*

(21) Application number: **10075688.1**

(22) Date of filing: **19.11.2004**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IS IT LI LU MC NL PL PT RO SE SI SK TR
Designated Extension States:
AL HR LT MK YU

(30) Priority: **20.11.2003 US 718318**

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
04798938.9 / 1 706 847

(71) Applicant: **ATI Technologies ULC**
Markham, Ontario L3T 7X6 (CA)

(72) Inventors:
- **Morein, Steven**
  Cambridge, Massachusetts 02139 (US)
- **Lefebvre, Laurent**
  Lachenale J6W BA5 (CA)
- **Gruber, Andy**
  Arlington, Massachusetts 02476 (US)
- **Skende, Andi**
  Shrewsbury, Massachusetts 01545 (US)

(74) Representative: **Waldren, Robin Michael**
Marks & Clerk LLP
90 Long Acre
London
WC2E 9RA (GB)

Remarks:
This application was filed on 01-10-2010 as a divisional application to the application mentioned under INID code 62.

(54) A graphics processing architecture employing a unified shader

(57) A graphics processor, comprising: an arbiter circuit for selecting one of a plurality of inputs in response to a control signal; a shader, coupled to the arbiter circuit, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and performing one of the vertex operations or pixel operations based on the selected one of the plurality of inputs, wherein the shader provides a appearance attribute; a vertex storage block for maintaining vertex information; wherein the vertex storage block further includes a parameter cache operative to maintain appearance attribute data for a corresponding vertex and a position cache operative to maintain position data for a corresponding vertex; and wherein the appearance attribute is color, and the color is associated with a corresponding pixel when the selected one of the plurality inputs is pixel data.

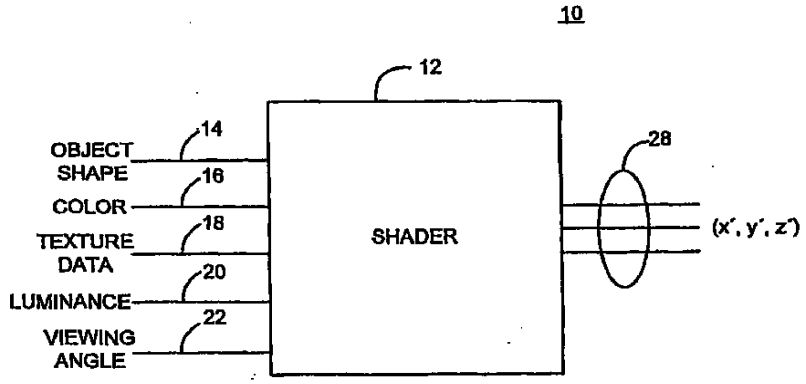


FIG. 1
(PRIOR ART)

EP 2 309 460 A1

Printed by Jouve, 75001 PARIS (FR)
TCL 1002

Description

FIELD OF THE INVENTION

[0001] The present invention generally relates to graphics processors and, more particularly, to a graphics processor architecture employing a single shader.

BACKGROUND OF THE INVENTION

[0002] In computer graphics applications, complex shapes and structures are formed through the sampling, interconnection and rendering of more simple objects, referred to as primitives. An example of such a primitive is a triangle, or other suitable polygon. These primitives, in turn, are formed by the interconnection of individual pixels. Color and texture are then applied to the individual pixels that comprise the shape based on their location within the primitive and the primitives orientation with respect to the generated shape; thereby generating the object that is rendered to a corresponding display for subsequent viewing.

[0003] The interconnection of primitives and the application of color and textures to generated shapes are generally performed by a graphics processor. Conventional graphics processors include a series of shaders that specify how and with what corresponding attributes, a final image is drawn on a screen, or suitable display device. As illustrated in FIG. 1, a conventional shader 10 can be represented as a processing block 12 that accepts a plurality of bits of input data, such as, for example, object shape data (14) in object space (x,y,z); material properties of the object, such as color (16); texture information (18); luminance information (20); and viewing angle information (22) and provides output data (28) representing the object with texture and other appearance properties applied thereto (x', y', z').

[0004] In exemplary fashion, as illustrated in FIGS. 2A-2B, the shader accepts the vertex coordinate data representing cube 30 (FIG. 2A) as inputs and provides data representing, for example, a perspectively corrected view of the cube 30' (FIG. 2B) as an output. The corrected view may be provided, for example, by applying an appropriate transformation matrix to the data representing the initial cube 30. More specifically, the representation illustrated in FIG. 2B is provided by a vertex shader that accepts as inputs the data representing, for example, vertices V_X, V_Y and V_Z, among others of cube 30 and providing angularly oriented vertices V_X,V_Y and V_Z, including any appearance attributes of corresponding cube 30'.

[0005] In addition to the vertex shader discussed above, a shader processing block that operates on the pixel level, referred to as a pixel shader is also used when generating an object for display. Generally, the pixel shader provides the color value associated with each pixel of a rendered object. Conventionally, both the vertex shader and pixel shader are separate components that are configured to perform only a single transformation or operation. Thus, in order to perform a position and a texture transformation of an input, at least two shading operations and hence, at least two shaders, need to be employed. Conventional graphics processors require the use of both a vertex shader and a pixel shader in order to generate an object. Because both types of shaders are required, known graphics processors are relatively large in size, with most of the real estate being taken up by the vertex and pixel shaders.

[0006] In addition to the real estate penalty associated with conventional graphics processors, there is also a corresponding performance penalty associated therewith. In conventional graphics processors, the vertex shader and the pixel shader are juxtaposed in a sequential, pipelined fashion, with the vertex shader being positioned before and operating on vertex data before the pixel shader can operate on individual pixel data.

[0007] Thus, there is a need for an improved graphics processor employing a shader that is both space efficient and computationally effective.

SUMMARY OF THE INVENTION

[0008] Briefly stated, the present invention is directed to a graphics processor that employs a unified shader that is capable of performing both the vertex operations and the pixel operations in a space saving and computationally efficient manner. In an exemplary embodiment, a graphics processor according to the present invention includes an arbiter circuit for selecting one of a plurality of inputs for processing in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs.

[0009] The shader includes a general purpose register block for storing at least the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs according to the instructions maintained in the sequencer. The shader of the present invention is referred to as a "unified" shader because it is configured to perform both vertex and pixel operations. By employing the unified shader of the present invention, the associated graphics processor is more space efficient than conventional graphics processors because the unified shader takes up less real estate than the conventional multi-shader processor architecture.

[0010] In addition, according to the present invention, the unified shader is more computationally efficient because it allows the shader to be flexibly allocated to pixels or vertices based on workload.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention and the associated advantages and features thereof, will become better understood and appreciated upon review of the following detailed description of the invention, taken in conjunction with the following drawings, where like numerals represent like elements, in which:

FIG. 1 is a schematic block diagram of a conventional shader;

FIGS. 2A-2B are graphical representations of the operations performed by the shader illustrated in FIG. 1;

FIG. 3 is a schematic block diagram of a conventional graphics processor architecture;

FIG. 4A is a schematic block diagram of a graphics processor architecture according to the present invention;

FIG. 4B is a schematic block diagram of an optional input component to the graphics processor according to an alternate embodiment of the present invention; and

FIG. 5 is an exploded schematic block diagram of the unified shader employed in the graphics processor illustrated in FIG. 4A.

DETAILED DESCRIPTION OF THE INVENTION

[0012] FIG. 3, illustrates a graphics processor incorporating a conventional pipeline architecture. As shown, the graphics processor 40 includes a vertex fetch block 42 which receives vertex information relating to a primitive to be rendered from an off-chip memory 55 on line 41. The fetched vertex data is then transmitted to a vertex cache 44 for storage on line 43. Upon request, the vertex data maintained in the vertex cache 44 is transmitted to a vertex shader 46 on line 45. As discussed above, an example of the information that is requested by and transmitted to the vertex shader 46 includes the object shape, material properties (e.g. color), texture information, and viewing angle. Generally, the vertex shader 46 is a programmable mechanism which applies a transformation position matrix to the input position information (obtained from the vertex cache 44), thereby providing data representing a perspectively corrected image of the object to be rendered, along with any texture or color coordinates thereof.

[0013] After performing the transformation operation, the data representing the transformed vertices are then provided to a vertex store 48 on line 47. The vertex store 48 then transmits the modified vertex information contained therein to a primitive assembly block 50 on line 49. The primitive assembly block 50 assembles, or converts, the input vertex information into a plurality of primitives to be subsequently processed. Suitable methods of assembling the input vertex information into primitives is known in the art and will not be discussed in greater detail here. The assembled primitives are then transmitted to a rasterization engine 52, which converts the previously assembled primitives into pixel data through a process referred to as walking. The resulting pixel data is then transmitted to a pixel shader 54 on line 53.

[0014] The pixel shader 54 generates the color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels. In addition, the pixel shader 54 is capable of fetching texture data from a texture map 57 as indexed by the pixel data from the rasterization engine 52 by transmitting such information on line 55 to the texture map. The requested texture data is then transmitted back from the texture map 57 on line 57' and stored in a texture cache 56 before being routed to the pixel shader on line 58. Once the texture data has been received, the pixel shader 54 then performs specified logical or arithmetic operations on the received texture data to generate the pixel color or other appearance attribute of interest. The generated pixel appearance attribute is then combined with a base color, as provided by the rasterization engine on line 53, to thereby provide a pixel color to the pixel corresponding at the position of interest. The pixel appearance attribute present on line 59 is then transmitted to post raster processing blocks (not shown).

[0015] As described above, the conventional graphics processor 40 requires the use of two separate shaders: a vertex shader 46 and a pixel shader 54. A drawback associated with such an architecture is that the overall footprint of the graphics processor is relatively large as the two shaders take up a large amount of real estate. Another drawback associated with conventional graphics processor architectures is that can exhibit poor computational efficiency.

[0016] Referring now to FIG. 4A, in an exemplary embodiment, the graphics processor 60 of the present invention includes a multiplexer 66 having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine 74 provided at a second input. A control signal generated by an arbiter 64 is transmitted to the multiplexer 66 on line 63. The arbiter 64 determines which of the two inputs to the multiplexer 66 is transmitted to a unified shader 62 for further processing. The arbitration scheme employed by the arbiter 64 is as follows: the vertex data on the first input of the multiplexer 66 is transmitted to the unified shader 62 on line 65 if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader 62 for further processing.

[0017] Referring briefly to FIG. 5, the unified shader

62 will now be described. As illustrated, the unified shader 62 includes a general purpose register block 92, a plurality of source registers: including source register A 93, source register B 95, and source register C 97, a processor (e.g. CPU) 96 and a sequencer 99. The general purpose register block 92 includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer 66 on line 65 or any other information to be maintained within the unified shader. The data present in the general purpose register block 92 is transmitted to the plurality of source registers via line 109.

[0018] The processor 96 may be comprised of a dedicated piece of hardware or can be configured as part of a general purpose computing device (i.e. personal computer). In an exemplary embodiment, the processor 96 is adapted to perform 32-bit floating point arithmetic operations as well as a complete series of logical operations on corresponding operands. As shown, the processor is logically partitioned into two sections. Section 96 is configured to execute, for example, the 32-bit floating point arithmetic operations of the unified shader. The second section, 96A, is configured to perform scaler operations (e.g. log, exponent, reciprocal square root) of the unified shader.

[0019] The sequencer 99 includes constants block 91 and an instruction store 98. The constants block 91 contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store 98 contains the necessary instructions that are executed by the processor 96 in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block 92 as provided by the source registers 93-95. The instruction store 98 further includes memory fetch instructions that, when executed, causes the unified shader 62 to fetch texture and other types of data, from memory 82 (FIG. 4A). In operation, the sequencer 99 determines whether the next instruction to be executed (from the instruction store 98) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer 99 sends the request to a fetch block (not shown) which retrieves the required information from memory 82 (FIG. 4A). The retrieved information is then transmitted to the sequencer 99, through the vertex texture cache 68 (FIG. 4A) as described in greater detail below.

[0020] If the next instruction to be executed is an arithmetic or logical instruction, the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block 92 into the appropriate source registers (93, 95, 97) for execution, and an appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor 96 executes the instructions on the operands present in the source registers and provides the result on line 85. The information present on line 85 may be transmitted back to the general purpose register block 92 for storage, or transmitted to succeeding components of the graphics processor 60.

[0021] As discussed above, the instruction store 98 maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader 99 of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader 62 of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.

[0022] The unified shader 62 has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store 98, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block 92 from multiplexer 66. The instruction store 98 then passes the corresponding control signals to the processor 96 on line 101 to perform such vertex operations. However, if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor 96 are continued, based on the instructions maintained in the instruction store 98, until enough registers within the general purpose register block 92 become available. Thus, through the sharing of resources within the unified shader 62, processing of image data is enhanced as there is no down time associated with the processor 96.

[0023] Referring back to FIG. 4A, the graphics processor 60 further includes a cache block 70, including a parameter cache 70A and a position cache 70B which accepts the pixel based output of the unified shader 62 on line 85 and stores the respective pixel parameter and position information in the corresponding cache. The pixel information present in the cache block 70 is then transmitted to the primitive assembly block 72 on line 71. The primitive assembly block 72 is responsible for assembling the information transmitted thereto from the cache block 70 into a series of triangles, or other suitable primitives, for further processing. The assembled primitives are then transmitted on line 73 to rasterization engine block 74, where the transmitted primitives are then converted into individual pixel data information through a walking process, or any other suitable pixel generation process. The resulting pixel data from the rasterization engine block 74 is the interpolated pixel parameter data that is transmitted to the second input of the multiplexer 66 on line 75.

[0024] In those situations when vertex data is transmitted to the unified shader 62 through the multiplexer 66, the resulting vertex data generated by the processor 96, is transmitted to a render back end block 76 which converts the resulting vertex data into at least one of several formats suitable for later display on display device 84. For example, if a stained glass appearance effect is to be applied to an image, the information corresponding to such appearance effect is associated with the appropriate position data by the render back end 76. the information from the render back end 76 is then transmitted to memory 82 and a display controller line 80 via memory controller 78. Such appropriately formatted information is then transmitted on line 83 for presentation on display device 84.

[0025] Referring now to FIG. 4B, shown therein is a vertex block 61 which is used to provide the vertex information at the first input of the multiplexer 66 according to an alternate embodiment of the present invention. The vertex block 61 includes a vertex fetch block 61A which is responsible for retrieving vertex information from memory 82, if requested, and transmitting that vertex information into the vertex cache 61 B. The information stored in the vertex cache 61 B comprises the vertex information that is coupled to the first input of multiplexer 66.

[0026] As discussed above, the graphics processor 60 of the present invention incorporates a unified shader 62 which is capable of performing both vertex manipulation operations and pixel manipulation operations based on the instructions stored in the instruction store 98. In this fashion, the graphics processor 60 of the present invention takes up less real estate than conventional graphics processors as separate vertex shaders and pixel shaders are no longer required. In addition, as the unified shader 62 is capable of alternating between performing vertex manipulation operations and pixel manipulation operations, graphics processing efficiency is enhanced as one type of data operations is not dependent upon another type of data operations. Therefore, any performance penalties experienced as a result of dependent operations in conventional graphics processors are overcome.

[0027] The above detailed description of the present invention and the examples described therein have been presented for the purposes of illustration and description. It is therefore contemplated that the present invention cover any and all modifications, variations and equivalents that fall within the scope of the basic underlying principles disclosed and claimed herein.

Claims

1. A graphics processor, comprising: an arbiter circuit for selecting one of a plurality of inputs in response to a control signal; a shader, coupled to the arbiter circuit, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and performing one of the vertex operations or pixel operations based on the selected one of the plurality of inputs, wherein the shader provides a appearance attribute; a vertex storage block for maintaining vertex information; wherein the vertex storage block further includes a parameter cache operative to maintain appearance attribute data for a corresponding vertex and a position cache operative to maintain position data for a corresponding vertex; and wherein the appearance attribute is color, and the color is associated with a corresponding pixel when the selected one of the plurality inputs is pixel data.

2. The graphics processor of claim 1 wherein the appearance attribute is position, and the position attribute is associated with a corresponding vertex when the selected one of the plurality of inputs is vertex data.

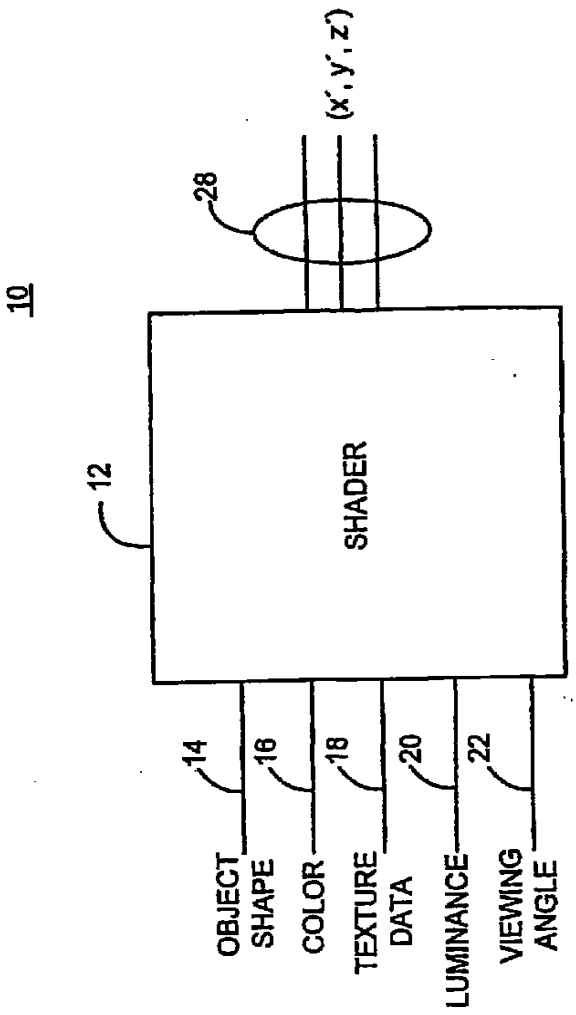


FIG. 1
(PRIOR ART)

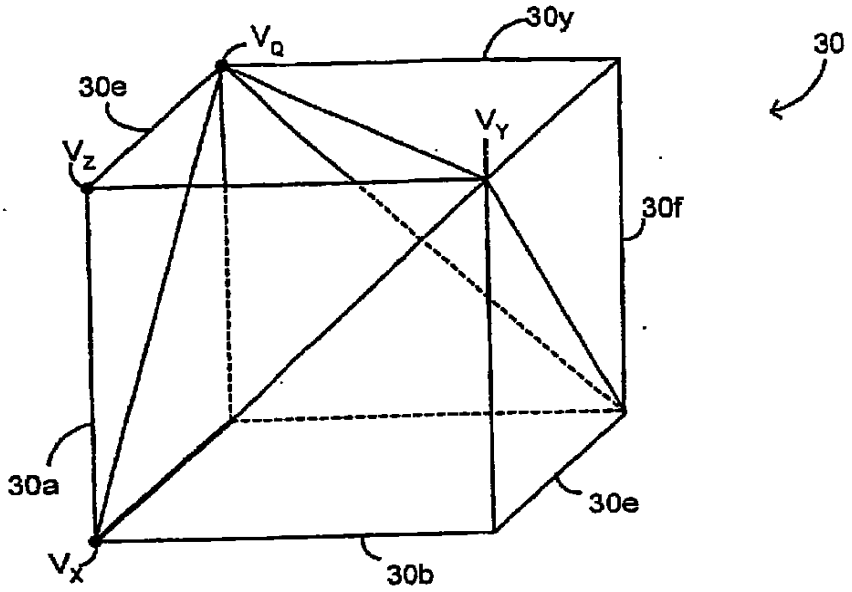


FIG. 2A
(PRIOR ART)

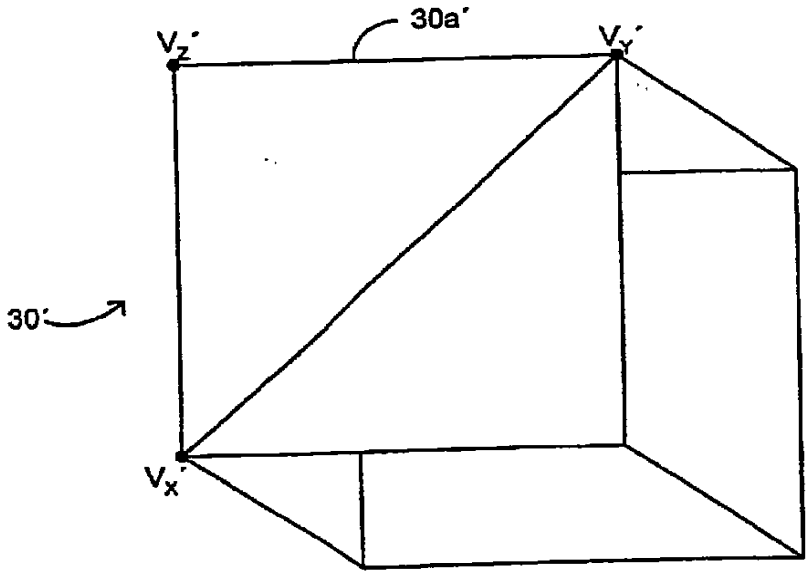


FIG. 2B
(PRIOR ART)

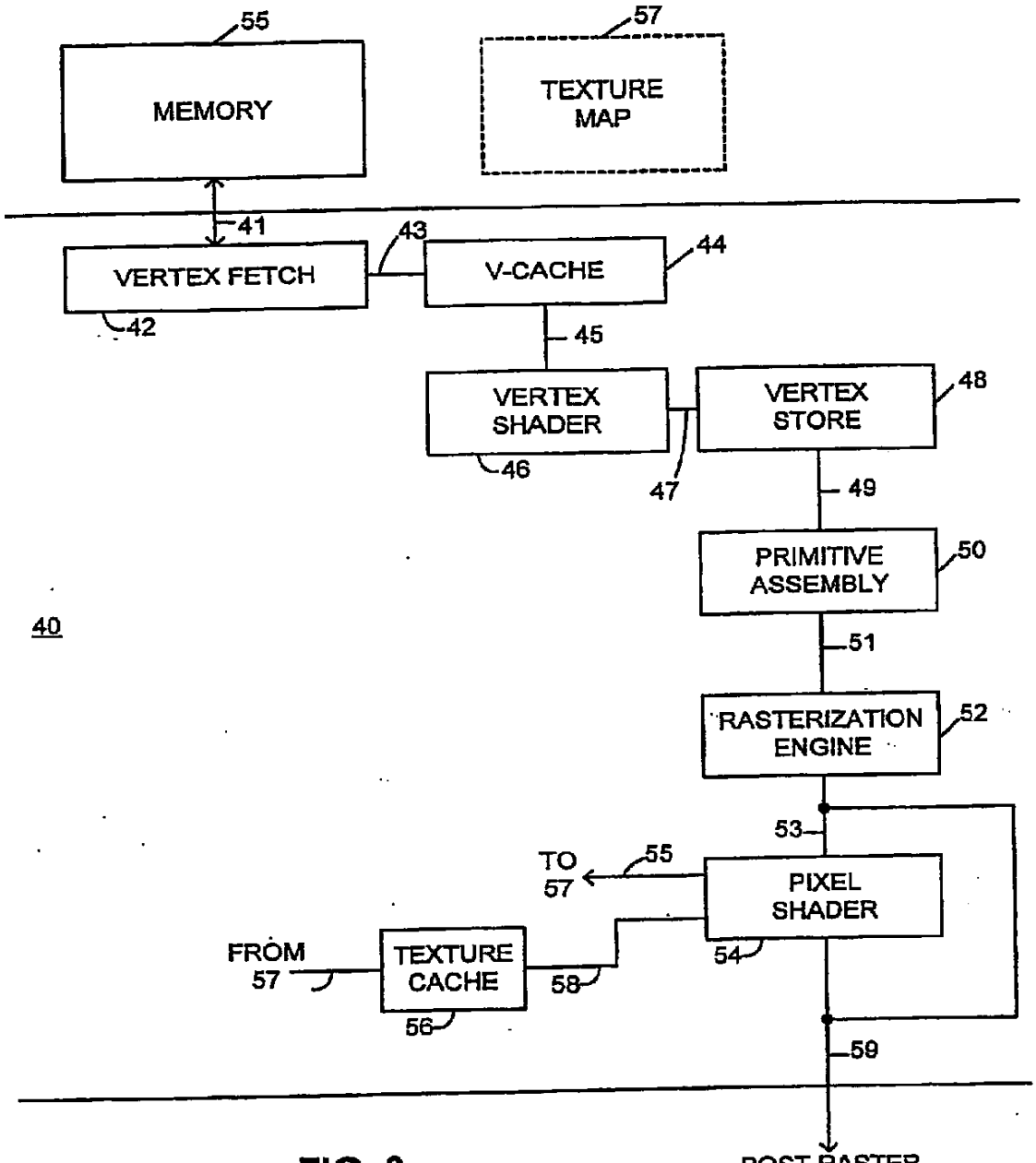


FIG. 3
(PRIOR ART)

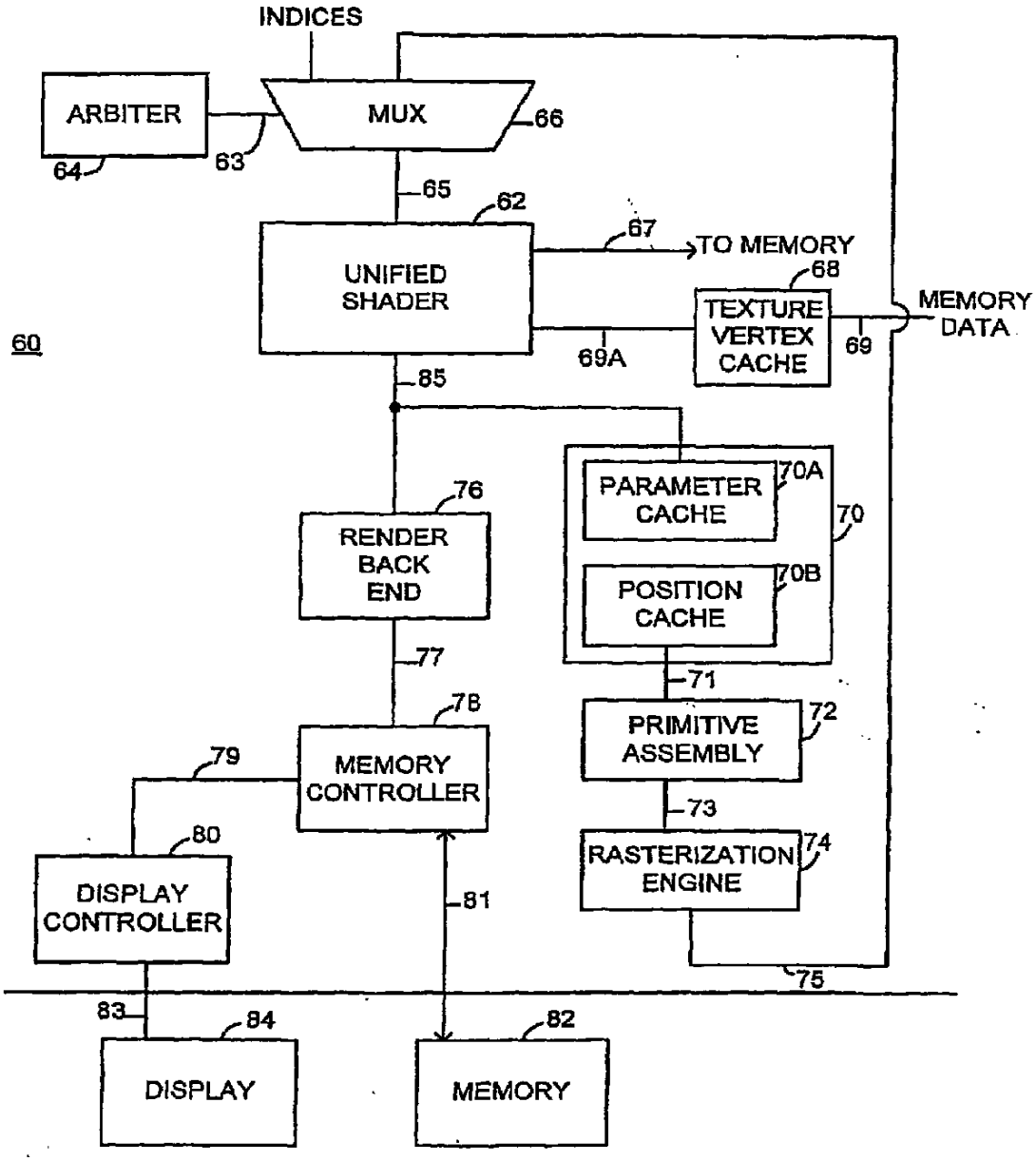

INDICES
ARBITER
MUX
66
64
63
65
62
67
TO MEMORY
UNIFIED SHADER
68
TEXTURE VERTEX CACHE
MEMORY DATA
60
69A
69
85
PARAMETER CACHE
70A
70
POSITION CACHE
70B
76
RENDER BACK END
77
71
78
PRIMITIVE ASSEMBLY
72
79
MEMORY CONTROLLER
73
80
RASTERIZATION ENGINE
74
DISPLAY CONTROLLER
81
75
83
84
82
DISPLAY
MEMORY

FIG. 4A

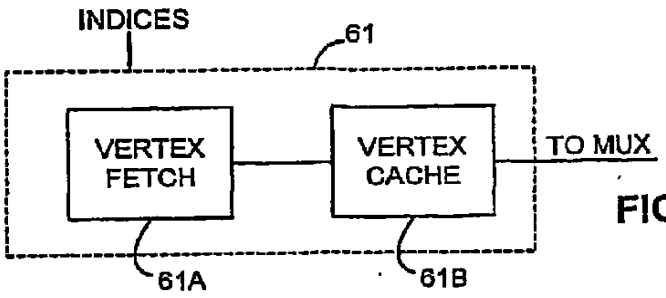

INDICES
61
VERTEX FETCH
VERTEX CACHE
TO MUX
61A
61B

FIG. 4B

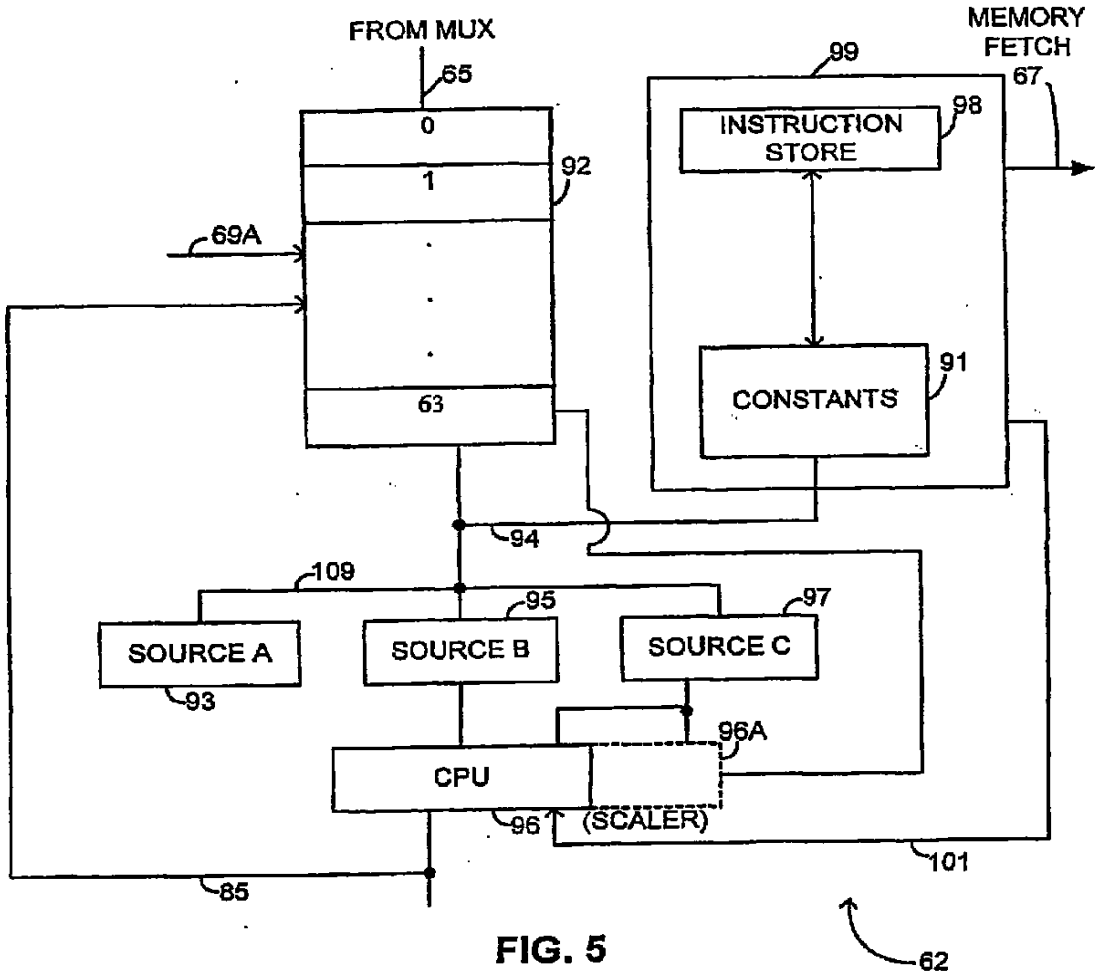

FROM MUX
65
0
1
63
92
69A
MEMORY FETCH
67
99
INSTRUCTION STORE
98
CONSTANTS
91
94
109
95
97
SOURCE A
SOURCE B
SOURCE C
93
96A
CPU
96
(SCALER)
101
85
62

FIG. 5

Europäisches Patentamt
European Patent Office
Office européen des brevets

EUROPEAN SEARCH REPORT

Application Number
EP 10 07 5688

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
A	US 2003/164830 A1 (KENT OSMAN) 4 September 2003 (2003-09-04) * abstract; figures 1A,1B,1C,1D * * paragraphs [0006], [0007], [0012] * * paragraphs [0079], [0091] - [0095], [0102], [0154] - [0156], [0170] * -----	1,2	INV. G06T15/00 G06T15/80
A	US 6 417 858 B1 (BOSCH DEREK ET AL) 9 July 2002 (2002-07-09) * abstract; figures 2,3,4,5 * * column 3, lines 22-32 * * column 8, line 47 - line 61 * * column 9, line 10 - line 21; claim 24 * -----	1,2	
A	US 6 353 439 B1 (LINDHOLM JOHN ERIK ET AL) 5 March 2002 (2002-03-05) * column 8, lines 22-53; figures 1B,2B,4,4B * -----	1,2	
A	BRETERNITZ M ET AL: "Compilation, architectural support,and evaluation of SIMD graphics pipeline programs on a general-purpose CPU", 27 September 2003 (2003-09-27), PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 2003. PACT 2003. PROCEEDINGS. 12TH INTERNATIONAL CONFERENCE ON 27 SEPT. - 1 OCT. 2003, PISCATAWAY, NJ, USA,IEEE, PAGE(S) 135-145, XP010662182, ISBN: 0-7695-2021-9 * page 1 - page 3; figures 1,2 * ----- -/--	1	TECHNICAL FIELDS SEARCHED (IPC) G06T

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
Munich	25 February 2011	Meinl, Wolfgang

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

2

Europäisches Patentamt
European Patent Office
Office européen des brevets

EUROPEAN SEARCH REPORT

Application Number
EP 10 07 5688

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
A	OWENS J D ET AL: "POLYGON RENDERING ON A STREAM ARCHITECTURE", PROCEEDINGS 2000 SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE. INTERLAKEN, SWITZERLAND, AUG. 21 - 22, 2000; [SIGGRAPH / EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE], NEW YORK, NY : ACM, US, 21 August 2000 (2000-08-21), pages 23-32, XP000964471, DOI: DOI:10.1145/346876.346883 ISBN: 978-1-58113-257-1 * abstract; figures 1,3 * * Sections 2, 2.1, 2.2, 3. * -----	1,2	
A	MARK W R ET AL: "Cg: a system for programming graphics hardware in a C-like language", ACM TRANSACTIONS ON GRAPHICS ACM USA, vol. 22, no. 3, July 2003 (2003-07), pages 896-907, XP002624786, ISSN: 0730-0301 * abstract; figure 2 * * page 899, column 1, lines 17-50 * -----	1,2	TECHNICAL FIELDS SEARCHED (IPC)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
Munich	25 February 2011	Meinl, Wolfgang

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 10 07 5688

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

25-02-2011

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003164830 A1	04-09-2003	NONE	
US 6417858 B1	09-07-2002	NONE	
US 6353439 B1	05-03-2002	NONE	

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 2 296 116 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
16.03.2011 Bulletin 2011/11

(51) Int Cl.:
G06T 15/00 *(2011.01)*

(21) Application number: **10075686.5**

(22) Date of filing: **19.11.2004**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IS IT LI LU MC NL PL PT RO SE SI SK TR
Designated Extension States:
AL HR LT MK YU

(30) Priority: **20.11.2003 US 718318**

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
04798938.9 / 1 706 847

(71) Applicant: **ATI Technologies Inc.**
Markham,
Ontario L3T 7X6 (CA)

(72) Inventors:
- **Morein, Steven**
  Cambridge, Massachusetts 02139 (US)
- **Lefebvre, Laurent**
  Lachenaie, Quebec J6W 6A5 (CA)
- **Gruber, Andy**
  Arlington, Massachusetts 02476 (US)
- **Skende, Andi**
  Shrewsbury, Massachusetts 01545 (US)

(74) Representative: **Waldren, Robin Michael**
Marks & Clerk LLP
90 Long Acre
London
WC2E 9RA (GB)

Remarks:
This application was filed on 01-10-2010 as a divisional application to the application mentioned under INID code 62.

(54) A graphics processing architecture employing a unified shader

(57) A method comprising:
performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

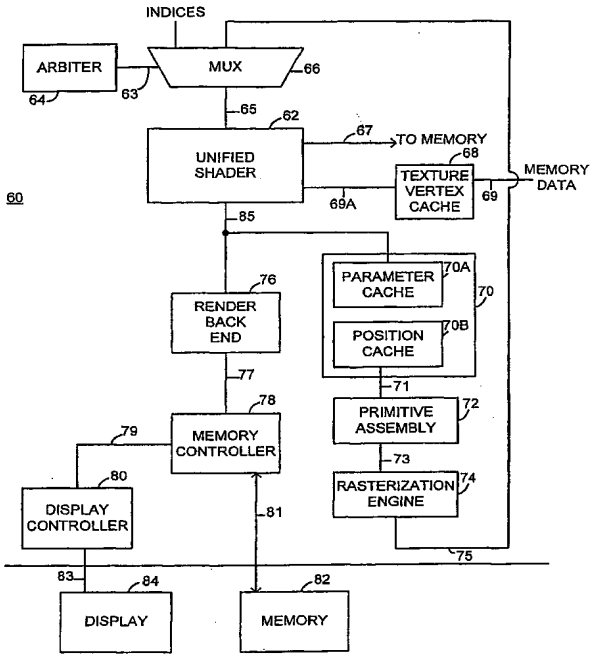


FIG. 4A

Description

FIELD OF THE INVENTION

[0001] The present invention generally relates to graphics processors and, more particularly, to a graphics processor architecture employing a single shader.

BACKGROUND OF THE INVENTION

[0002] In computer graphics applications, complex shapes and structures are formed through the sampling, interconnection and rendering of more simple objects, referred to as primitives. An example of such a primitive is a triangle, or other suitable polygon. These primitives, in turn, are formed by the interconnection of individual pixels. Color and texture are then applied to the individual pixels that comprise the shape based on their location within the primitive and the primitives orientation with respect to the generated shape; thereby generating the object that is rendered to a corresponding display for subsequent viewing.

[0003] The interconnection of primitives and the application of color and textures to generated shapes are generally performed by a graphics processor. Conventional graphics processors include a series of shaders that specify how and with what corresponding attributes, a final image is drawn on a screen, or suitable display device. As illustrated in FIG. 1, a conventional shader 10 can be represented as a processing block 12 that accepts a plurality of bits of input data, such as, for example, object shape data (14) in object space (x,y,z); material properties of the object, such as color (16); texture information (18); luminance information (20); and viewing angle information (22) and provides output data (28) representing the object with texture and other appearance properties applied thereto (x', y', z').

[0004] In exemplary fashion, as illustrated in FIGS. 2A-2B, the shader accepts the vertex coordinate data representing cube 30 (FIG. 2A) as inputs and provides data representing, for example, a perspectively corrected view of the cube 30' (FIG. 2B) as an output. The corrected view may be provided, for example, by applying an appropriate transformation matrix to the data representing the initial cube 30. More specifically, the representation illustrated in FIG. 2B is provided by a vertex shader that accepts as inputs the data representing, for example, vertices V_X, V_Y and V_Z, among others of cube 30 and providing angularly oriented vertices $V_{X'}$, $V_{Y'}$ and $V_{Z'}$, including any appearance attributes of corresponding cube 30'.

[0005] In addition to the vertex shader discussed above, a shader processing block that operates on the pixel level, referred to as a pixel shader is also used when generating an object for display. Generally, the pixel shader provides the color value associated with each pixel of a rendered object. Conventionally, both the vertex shader and pixel shader are separate components that are configured to perform only a single transformation or operation. Thus, in order to perform a position and a texture transformation of an input, at least two shading operations and hence, at least two shaders, need to be employed. Conventional graphics processors require the use of both a vertex shader and a pixel shader in order to generate an object. Because both types of shaders are required, known graphics processors are relatively large in size, with most of the real estate being taken up by the vertex and pixel shaders.

[0006] In addition to the real estate penalty associated with conventional graphics processors, there is also a corresponding performance penalty associated therewith. In conventional graphics processors, the vertex shader and the pixel shader are juxtaposed in a sequential, pipelined fashion, with the vertex shader being positioned before and operating on vertex data before the pixel shader can operate on individual pixel data.

[0007] Thus, there is a need for an improved graphics processor employing a shader that is both space efficient and computationally effective.

SUMMARY OF THE INVENTION

[0008] Briefly stated, the present invention is directed to a graphics processor that employs a unified shader that is capable of performing both the vertex operations and the pixel operations in a space saving and computationally efficient manner. In an exemplary embodiment, a graphics processor according to the present invention includes an arbiter circuit for selecting one of a plurality of inputs for processing in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs.

[0009] The shader includes a general purpose register block for storing at least the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs according to the instructions maintained in the sequencer. The shader of the present invention is referred to as a "unified" shader because it is configured to perform both vertex and pixel operations. By employing the unified shader of the present invention, the associated graphics processor is more space efficient than conventional graphics processors because the unified shader takes up less real estate than the conventional multi-shader processor architecture.

[0010] In addition, according to the present invention, the unified shader is more computationally efficient because it allows the shader to be flexibly allocated to pixels or vertices based on workload.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention and the associated advantages and features thereof, will become better understood and appreciated upon review of the following detailed description of the invention, taken in conjunction with the following drawings, where like numerals represent like elements, in which:

FIG. 1 is a schematic block diagram of a conventional shader;

FIGS. 2A-2B are graphical representations of the operations performed by the shader illustrated in FIG. 1;

FIG. 3 is a schematic block diagram of a conventional graphics processor architecture;

FIG. 4A is a schematic block diagram of a graphics processor architecture according to the present invention;

FIG. 4B is a schematic block diagram of an optional input component to the graphics processor according to an alternate embodiment of the present invention; and

FIG. 5 is an exploded schematic block diagram of the unified shader employed in the graphics processor illustrated in FIG. 4A.

DETAILED DESCRIPTION OF THE INVENTION

[0012] FIG. 3, illustrates a graphics processor incorporating a conventional pipeline architecture. As shown, the graphics processor 40 includes a vertex fetch block 42 which receives vertex information relating to a primitive to be rendered from an off-chip memory 55 on line 41. The fetched vertex data is then transmitted to a vertex cache 44 for storage on line 43. Upon request, the vertex data maintained in the vertex cache 44 is transmitted to a vertex shader 46 on line 45. As discussed above, an example of the information that is requested by and transmitted to the vertex shader 46 includes the object shape, material properties (e.g. color), texture information, and viewing angle. Generally, the vertex shader 46 is a programmable mechanism which applies a transformation position matrix to the input position information (obtained from the vertex cache 44), thereby providing data representing a perspectively corrected image of the object to be rendered, along with any texture or color coordinates thereof.

[0013] After performing the transformation operation, the data representing the transformed vertices are then provided to a vertex store 48 on line 47. The vertex store 48 then transmits the modified vertex information contained therein to a primitive assembly block 50 on line 49. The primitive assembly block 50 assembles, or converts, the input vertex information into a plurality of primitives to be subsequently processed. Suitable methods of assembling the input vertex information into primitives is known in the art and will not be discussed in greater detail here. The assembled primitives are then transmitted to a rasterization engine 52, which converts the previously assembled primitives into pixel data through a process referred to as walking. The resulting pixel data is then transmitted to a pixel shader 54 on line 53.

[0014] The pixel shader 54 generates the color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels. In addition, the pixel shader 54 is capable of fetching texture data from a texture map 57 as indexed by the pixel data from the rasterization engine 52 by transmitting such information on line 55 to the texture map. The requested texture data is then transmitted back from the texture map 57 on line 57' and stored in a texture cache 56 before being routed to the pixel shader on line 58. Once the texture data has been received, the pixel shader 54 then performs specified logical or arithmetic operations on the received texture data to generate the pixel color or other appearance attribute of interest. The generated pixel appearance attribute is then combined with a base color, as provided by the rasterization engine on line 53, to thereby provide a pixel color to the pixel corresponding at the position of interest. The pixel appearance attribute present on line 59 is then transmitted to post raster processing blocks (not shown).

[0015] As described above, the conventional graphics processor 40 requires the use of two separate shaders: a vertex shader 46 and a pixel shader 54. A drawback associated with such an architecture is that the overall footprint of the graphics processor is relatively large as the two shaders take up a large amount of real estate. Another drawback associated with conventional graphics processor architectures is that can exhibit poor computational efficiency.

[0016] Referring now to FIG. 4A, in an exemplary embodiment, the graphics processor 60 of the present invention includes a multiplexer 66 having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine 74 provided at a second input. A control signal generated by an arbiter 64 is transmitted to the multiplexer 66 on line 63. The arbiter 64 determines which of the two inputs to the multiplexer 66 is transmitted to a unified shader 62 for further processing. The arbitration scheme employed by the arbiter 64 is as follows: the vertex data on the first input of the multiplexer 66 is transmitted to the unified shader 62 on line 65 if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader 62 for further processing.

[0017] Referring briefly to FIG. 5, the unified shader

62 will now be described. As illustrated, the unified shader 62 includes a general purpose register block 92, a plurality of source registers: including source register A 93, source register B 95, and source register C 97, a processor (e.g. CPU) 96 and a sequencer 99. The general purpose register block 92 includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer 66 on line 65 or any other information to be maintained within the unified shader. The data present in the general purpose register block 92 is transmitted to the plurality of source registers via line 109.

[0018] The processor 96 may be comprised of a dedicated piece of hardware or can be configured as part of a general purpose computing device (i.e. personal computer). In an exemplary embodiment, the processor 96 is adapted to perform 32-bit floating point arithmetic operations as well as a complete series of logical operations on corresponding operands. As shown, the processor is logically partitioned into two sections. Section 96 is configured to execute, for example, the 32-bit floating point arithmetic operations of the unified shader. The second section, 96A, is configured to perform scaler operations (e.g. log, exponent, reciprocal square root) of the unified shader.

[0019] The sequencer 99 includes constants block 91 and an instruction store 98. The constants block 91 contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store 98 contains the necessary instructions that are executed by the processor 96 in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block 92 as provided by the source registers 93-95. The instruction store 98 further includes memory fetch instructions that, when executed, causes the unified shader 62 to fetch texture and other types of data, from memory 82 (FIG. 4A). In operation, the sequencer 99 determines whether the next instruction to be executed (from the instruction store 98) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer 99 sends the request to a fetch block (not shown) which retrieves the required information from memory 82 (FIG. 4A). The retrieved information is then transmitted to the sequencer 99, through the vertex texture cache 68 (FIG. 4A) as described in greater detail below.

[0020] If the next instruction to be executed is an arithmetic or logical instruction, the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block 92 into the appropriate source registers (93, 95, 97) for execution, and an appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor 96 executes the instructions on the operands present in the source registers and provides the result on line 85. The information present on line 85 may be transmitted back to the general purpose register block 92 for storage, or transmitted to succeeding components of the graphics processor 60.

[0021] As discussed above, the instruction store 98 maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader 99 of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader 62 of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.

[0022] The unified shader 62 has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store 98, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block 92 from multiplexer 66. The instruction store 98 then passes the corresponding control signals to the processor 96 on line 101 to perform such vertex operations. However, if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor 96 are continued, based on the instructions maintained in the instruction store 98, until enough registers within the general purpose register block 92 become available. Thus, through the sharing of resources within the unified shader 62, processing of image data is enhanced as there is no down time associated with the processor 96.

[0023] Referring back to FIG. 4A, the graphics processor 60 further includes a cache block 70, including a parameter cache 70A and a position cache 70B which accepts the pixel based output of the unified shader 62 on line 85 and stores the respective pixel parameter and position information in the corresponding cache. The pixel information present in the cache block 70 is then transmitted to the primitive assembly block 72 on line 71. The primitive assembly block 72 is responsible for assembling the information transmitted thereto from the cache block 70 into a series of triangles, or other suitable primitives, for further processing. The assembled primitives are then transmitted on line 73 to rasterization engine block 74, where the transmitted primitives are then converted into individual pixel data information through a walking process, or any other suitable pixel generation process. The resulting pixel data from the rasterization engine block 74 is the interpolated pixel parameter data that is transmitted to the second input of the multiplexer 66 on line 75.

[0024] In those situations when vertex data is transmitted to the unified shader 62 through the multiplexer 66, the resulting vertex data generated by the processor 96, is transmitted to a render back end block 76 which converts the resulting vertex data into at least one of several formats suitable for later display on display device 84. For example, if a stained glass appearance effect is to be applied to an image, the information corresponding to such appearance effect is associated with the appropriate position data by the render back end 76. The information from the render back end 76 is then transmitted to memory 82 and a display controller line 80 via memory controller 78. Such appropriately formatted information is then transmitted on line 83 for presentation on display device 84.

[0025] Referring now to FIG. 4B, shown therein is a vertex block 61 which is used to provide the vertex information at the first input of the multiplexer 66 according to an alternate embodiment of the present invention. The vertex block 61 includes a vertex fetch block 61A which is responsible for retrieving vertex information from memory 82, if requested, and transmitting that vertex information into the vertex cache 61 B. The information stored in the vertex cache 61 B comprises the vertex information that is coupled to the first input of multiplexer 66.

[0026] As discussed above, the graphics processor 60 of the present invention incorporates a unified shader 62 which is capable of performing both vertex manipulation operations and pixel manipulation operations based on the instructions stored in the instruction store 98. In this fashion, the graphics processor 60 of the present invention takes up less real estate than conventional graphics processors as separate vertex shaders and pixel shaders are no longer required. In addition, as the unified shader 62 is capable of alternating between performing vertex manipulation operations and pixel manipulation operations, graphics processing efficiency is enhanced as one type of data operations is not dependent upon another type of data operations. Therefore, any performance penalties experienced as a result of dependent operations in conventional graphics processors are overcome.

[0027] The above detailed description of the present invention and the examples described therein have been presented for the purposes of illustration and description. It is therefore contemplated that the present invention cover any and all modifications, variations and equivalents that fall within the scope of the basic underlying principles disclosed and claimed herein.

Claims

1. A method comprising:

   performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

2. A unified shader, comprising:

   a general purpose register block for maintaining data;
   a processor unit operative to:

   perform vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and perform vertex operations on the vertex data unless the general purpose register block does not have enough available space therein to store incoming vertex data and continue pixel calculation operations that are to be or are currently being performed based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

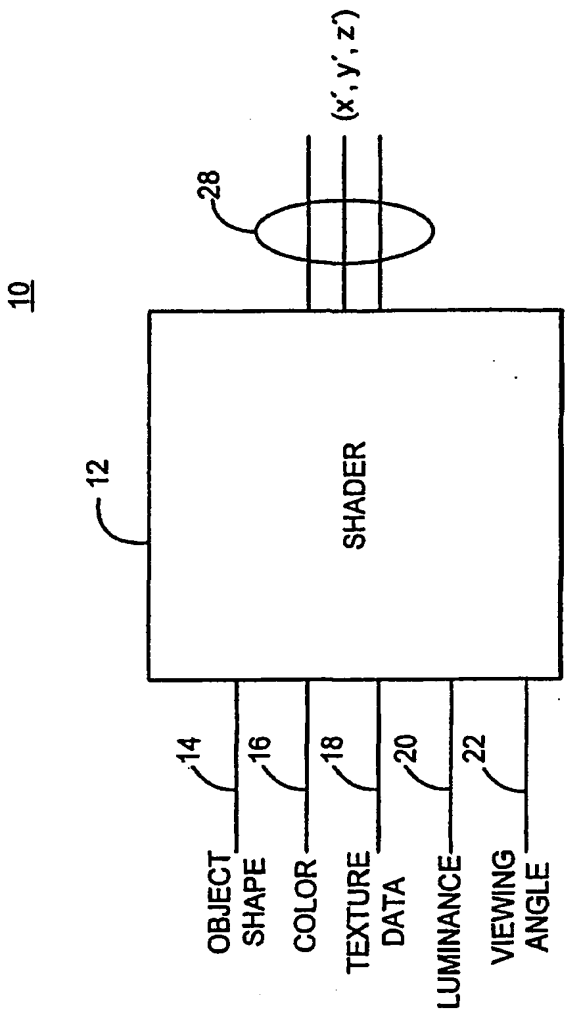


FIG. 1
(PRIOR ART)

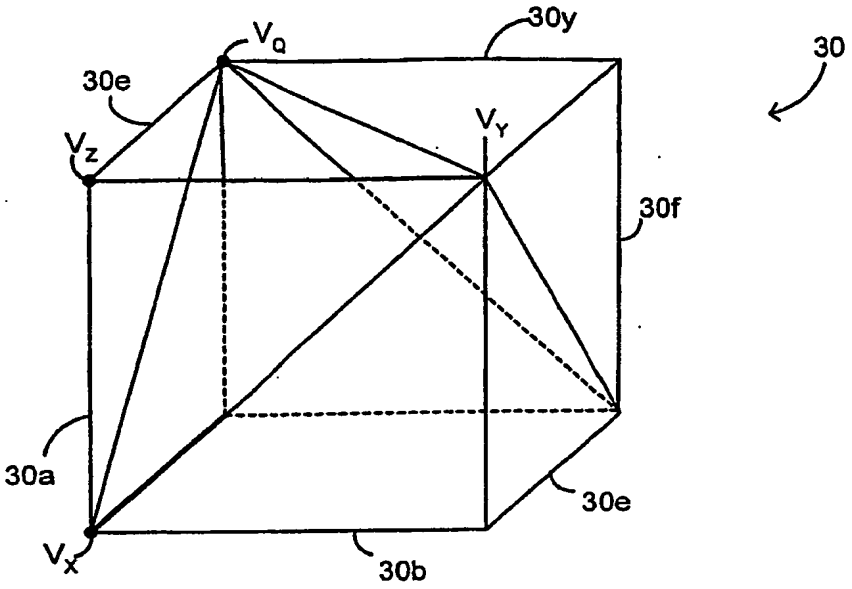


FIG. 2A
(PRIOR ART)

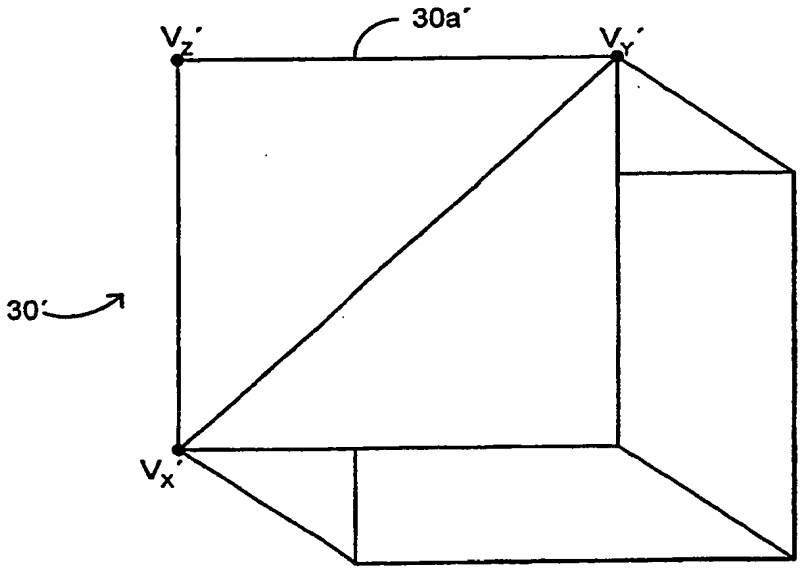


FIG. 2B
(PRIOR ART)

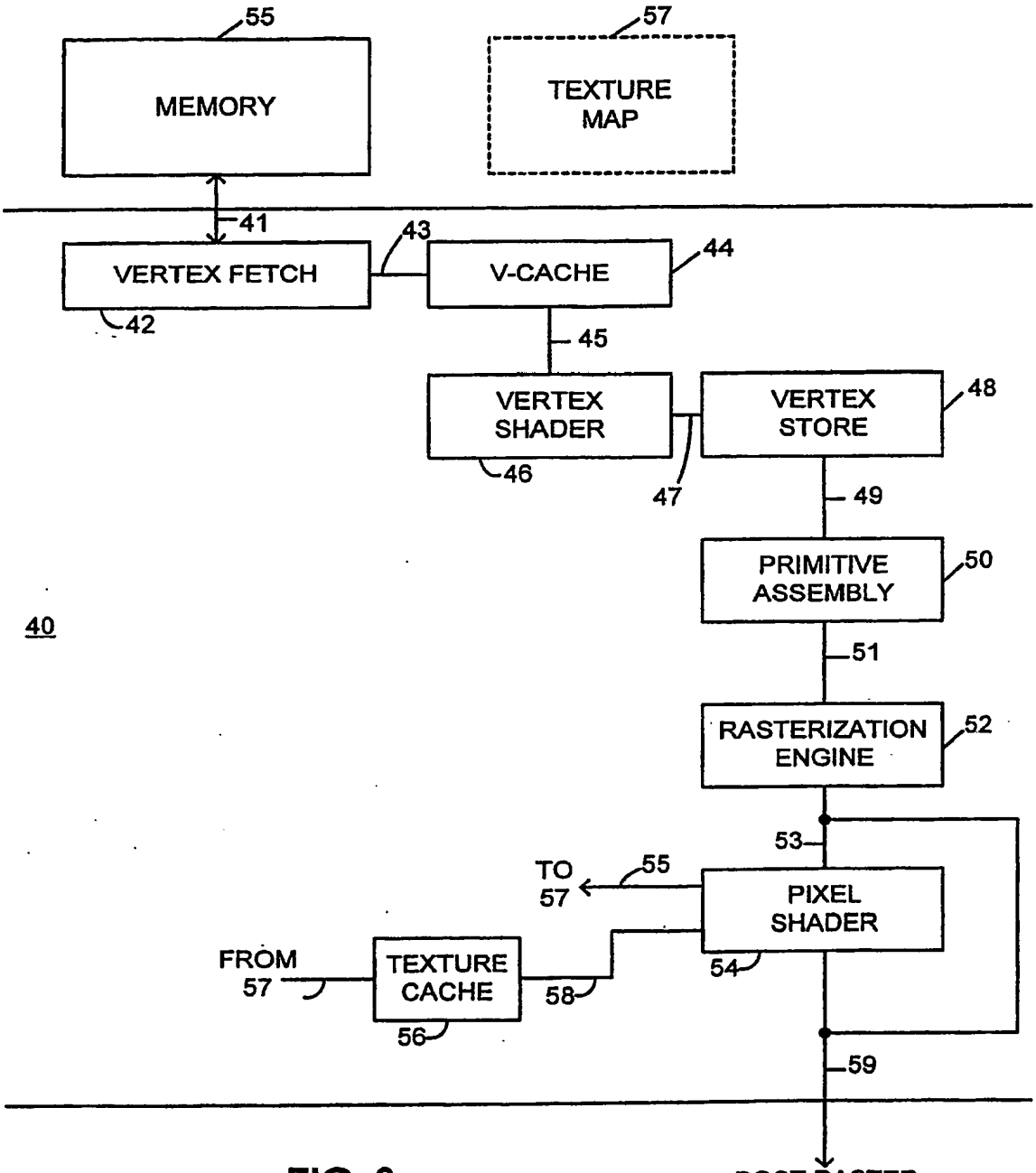


FIG. 3
(PRIOR ART)

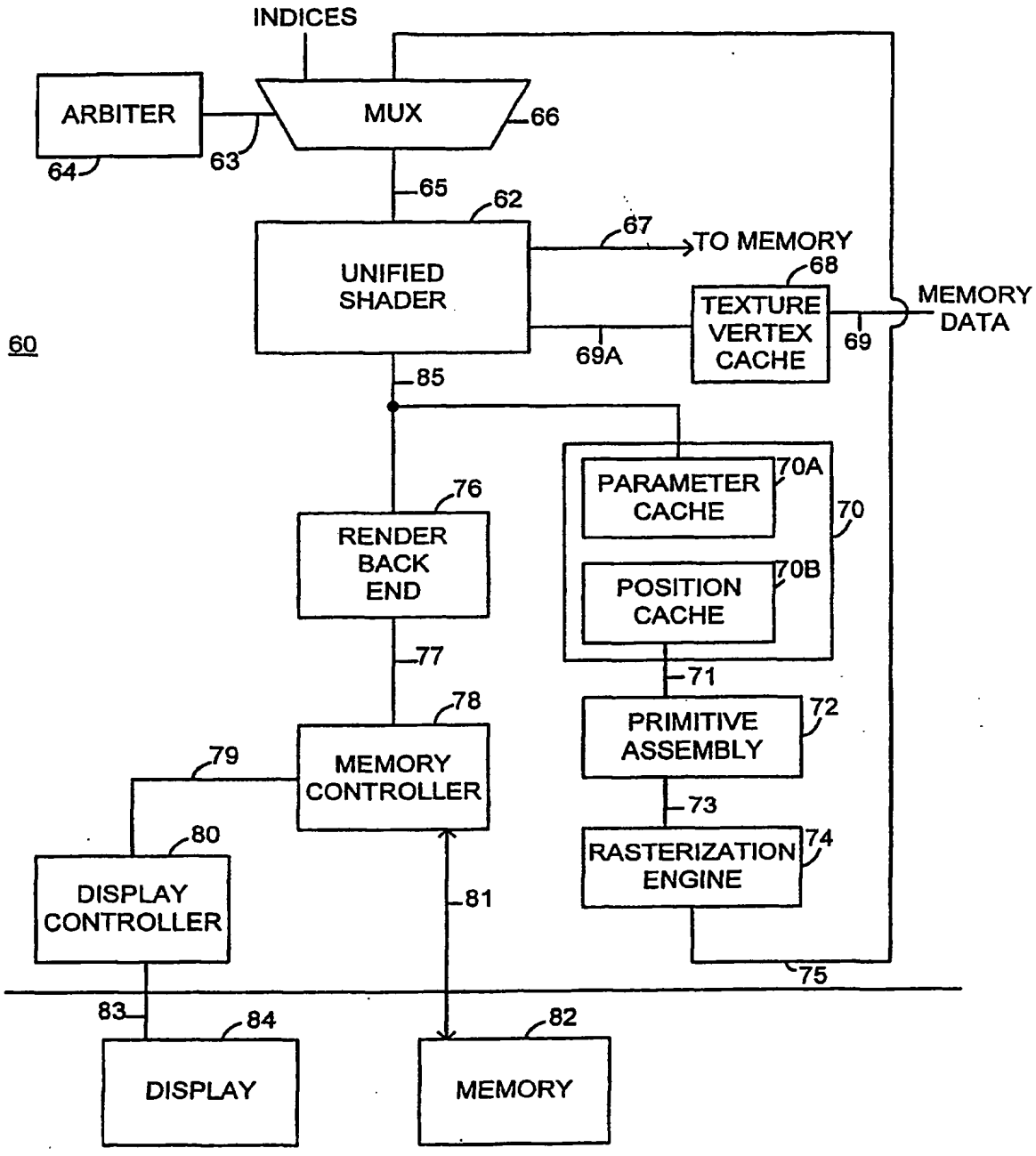

INDICES
ARBITER
MUX
66
64
63
65
62
67
TO MEMORY
UNIFIED SHADER
68
TEXTURE VERTEX CACHE
MEMORY DATA
60
69A
69
85
PARAMETER CACHE
70A
70
RENDER BACK END
76
POSITION CACHE
70B
77
71
78
PRIMITIVE ASSEMBLY
72
79
MEMORY CONTROLLER
73
80
RASTERIZATION ENGINE
74
DISPLAY CONTROLLER
81
75
83
84
82
DISPLAY
MEMORY

FIG. 4A

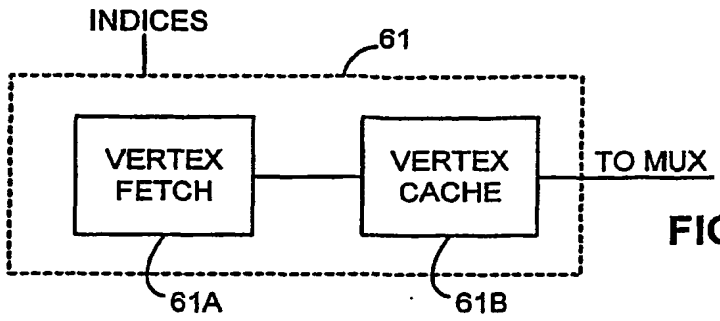

INDICES
61
VERTEX FETCH
VERTEX CACHE
TO MUX
61A
61B

FIG. 4B

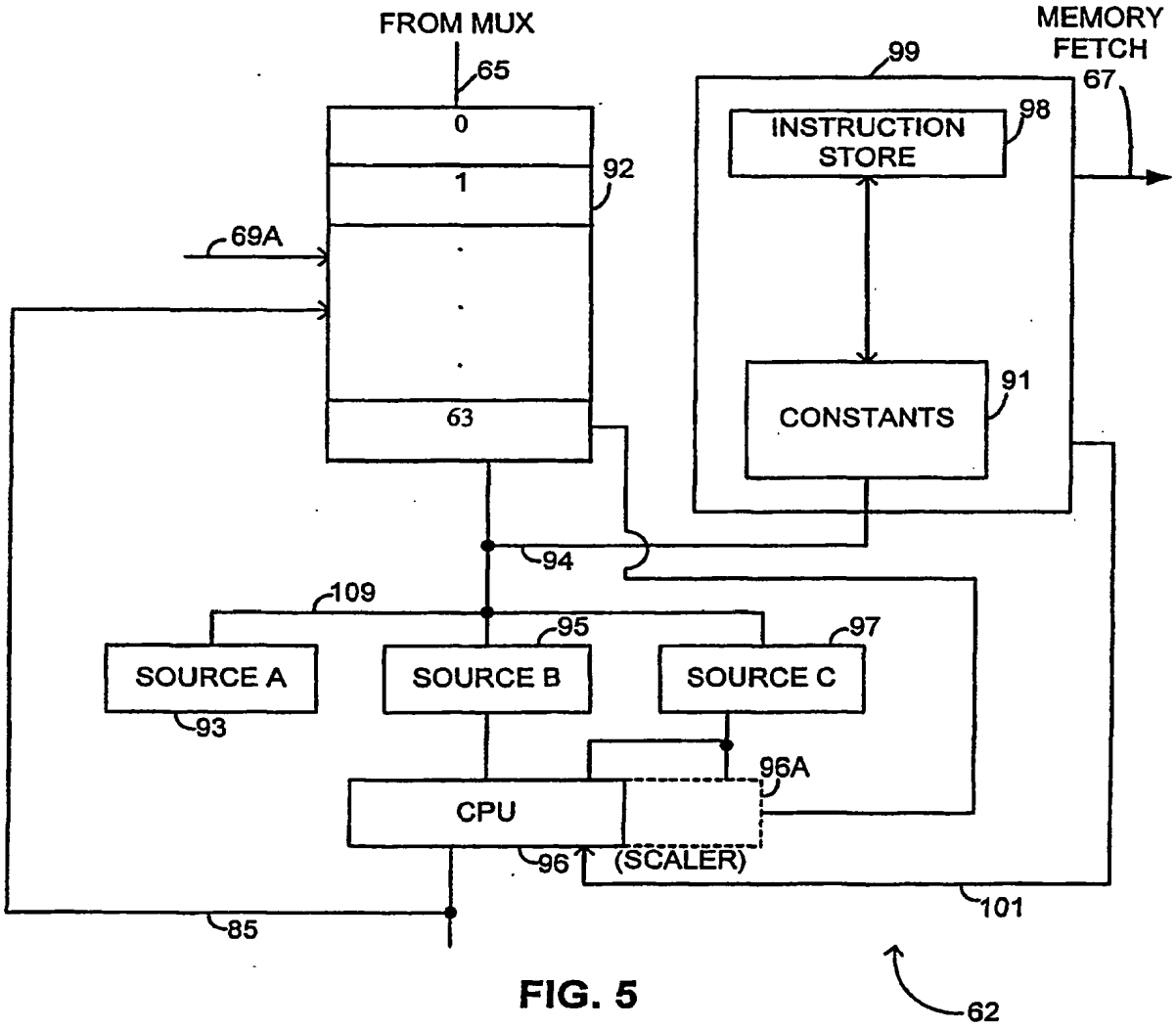

FROM MUX
65
0
1
63
92
69A
MEMORY FETCH
67
99
INSTRUCTION STORE
98
CONSTANTS
91
94
109
95
97
SOURCE A
SOURCE B
SOURCE C
93
96A
CPU
96
(SCALER)
101
85
62

FIG. 5

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference	FOR FURTHER ACTION	see Form PCT/ISA/220 as well as, where applicable, item 5 below.
SH-47386		
International application No.	International filing date *(day/month/year)*	(Earliest) Priority Date *(day/month/year)*
PCT/IB2004/003821	19/11/2004	20/11/2003
Applicant		
ATI TECHNOLOGIES, INC		

This International Search Report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This International Search Report consists of a total of 4 sheets.

[X] It is also accompanied by a copy of each prior art document cited in this report.

1. **Basis of the report**
   a. With regard to the **language**, the international search was carried out on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.
      [ ] The international search was carried out on the basis of a translation of the international application furnished to this Authority (Rule 23.1(b)).
   b. [ ] With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, see Box No. I.

2. [ ] **Certain claims were found unsearchable** (See Box II).

3. [ ] **Unity of invention is lacking** (see Box III).

4. With regard to the **title**,
   [X] the text is approved as submitted by the applicant.
   [ ] the text has been established by this Authority to read as follows:

5. With regard to the **abstract**,
   [X] the text is approved as submitted by the applicant.
   [ ] the text has been established, according to Rule 38.2(b), by this Authority as it appears in Box No. IV. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.

6. With regard to the **drawings**,
   a. the figure of the **drawings** to be published with the abstract is Figure No. 4A
      [X] as suggested by the applicant.
      [ ] as selected by this Authority, because the applicant failed to suggest a figure.
      [ ] as selected by this Authority, because this figure better characterizes the invention.
   b. [ ] none of the figures is to be published with the abstract.

Form PCT/ISA/210 (first sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB2004/003821

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06T15/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06T

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2003/164830 A1 (KENT OSMAN) 4 September 2003 (2003-09-04) paragraphs '0079!, '0091! - '0095!, '0102!, '0154! - '0156!, '0170!	1-13
A	US 6 417 858 B1 (BOSCH DEREK ET AL) 9 July 2002 (2002-07-09) column 8, line 47 - line 61 column 9, line 10 - line 21; claim 24; figure 5	14-20
A	US 6 353 439 B1 (LINDHOLM JOHN ERIK ET AL) 5 March 2002 (2002-03-05) column 26, line 16 - line 65; figures 23,24	14-20

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 March 2005	22/03/2005

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax. (+31-70) 340-3016	Tibaux, M

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB2004/003821

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	BRETERNITZ M ET AL: "Compilation, architectural support,and evaluation of SIMD graphics pipeline programs on a general-purpose CPU" 27 September 2003 (2003-09-27), PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 2003. PACT 2003. PROCEEDINGS. 12TH INTERNATIONAL CONFERENCE ON 27 SEPT. - 1 OCT. 2003, PISCATAWAY, NJ, USA,IEEE, PAGE(S) 135-145 , XP010662182 ISBN: 0-7695-2021-9 page 2, left-hand column, paragraph 4 - page 3, right-hand column, paragraph 4	1-20

Form PCT/ISA/210 (continuation of second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/IB2004/003821

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003164830 A1	04-09-2003	NONE	
US 6417858 B1	09-07-2002	NONE	
US 6353439 B1	05-03-2002	US 6198488 B1	06-03-2001
		AU 1948501 A	12-06-2001
		CA 2392371 A1	07-06-2001
		EP 1261939 A1	04-12-2002
		JP 2003515851 T	07-05-2003
		WO 0141069 A1	07-06-2001
		US 2003112246 A1	19-06-2003
		US 2001017626 A1	30-08-2001
		US 6844880 B1	18-01-2005
		AU 2064501 A	12-06-2001
		CA 2392370 A1	07-06-2001
		EP 1238371 A1	11-09-2002
		JP 2003515853 T	07-05-2003
		WO 0141073 A1	07-06-2001
		US 2002196259 A1	26-12-2002
		US 2002180740 A1	05-12-2002
		US 2003112245 A1	19-06-2003
		US 2003103054 A1	05-06-2003
		US 2003189565 A1	09-10-2003
		US 6452595 B1	17-09-2002
		US 6342888 B1	29-01-2002
		US 2001005209 A1	28-06-2001
		US 2002105519 A1	08-08-2002
		US 2003103050 A1	05-06-2003
		US 2002027553 A1	07-03-2002
		US 2002047846 A1	25-04-2002

PATENT COOPERATION TREATY

From the
INTERNATIONAL SEARCHING AUTHORITY

To:

see form PCT/ISA/220

PCT

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

(PCT Rule 43*bis*.1)

Date of mailing
(day/month/year) see form PCT/ISA/210 (second sheet)

Applicant's or agent's file reference
see form PCT/ISA/220

FOR FURTHER ACTION
See paragraph 2 below

International application No.	International filing date *(day/month/year)*	Priority date *(day/month/year)*
PCT/IB2004/003821	19.11.2004	20.11.2003

International Patent Classification (IPC) or both national classification and IPC
G06T15/00

Applicant
ATI TECHNOLOGIES, INC

1. This opinion contains indications relating to the following items:

- ☒ Box No. I Basis of the opinion
- ☐ Box No. II Priority
- ☐ Box No. III Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
- ☐ Box No. IV Lack of unity of invention
- ☒ Box No. V Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
- ☐ Box No. VI Certain documents cited
- ☐ Box No. VII Certain defects in the international application
- ☐ Box No. VIII Certain observations on the international application

2. **FURTHER ACTION**

If a demand for international preliminary examination is made, this opinion will usually be considered to be a written opinion of the International Preliminary Examining Authority ("IPEA"). However, this does not apply where the applicant chooses an Authority other than this one to be the IPEA and the chosen IPEA has notified the International Bureau under Rule 66.1*bis*(b) that written opinions of this International Searching Authority will not be so considered.

If this opinion is, as provided above, considered to be a written opinion of the IPEA, the applicant is invited to submit to the IPEA a written reply together, where appropriate, with amendments, before the expiration of three months from the date of mailing of Form PCT/ISA/220 or before the expiration of 22 months from the priority date, whichever expires later.

For further options, see Form PCT/ISA/220.

3. For further details, see notes to Form PCT/ISA/220.

Name and mailing address of the ISA:

European Patent Office
D-80298 Munich
Tel. +49 89 2399 - 0 Tx: 523656 epmu d
Fax: +49 89 2399 - 4465

Authorized Officer

Tibaux, M

Telephone No. +49 89 2399-2656

Box No. I Basis of the opinion

1. With regard to the **language**, this opinion has been established on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

   ☐ This opinion has been established on the basis of a translation from the original language into the following language , which is the language of a translation furnished for the purposes of international search (under Rules 12.3 and 23.1(b)).

2. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application and necessary to the claimed invention, this opinion has been established on the basis of:

   a. type of material:

   ☐ a sequence listing

   ☐ table(s) related to the sequence listing

   b. format of material:

   ☐ in written format

   ☐ in computer readable form

   c. time of filing/furnishing:

   ☐ contained in the international application as filed.

   ☐ filed together with the international application in computer readable form.

   ☐ furnished subsequently to this Authority for the purposes of search.

3. ☐ In addition, in the case that more than one version or copy of a sequence listing and/or table relating thereto has been filed or furnished, the required statements that the information in the subsequent or additional copies is identical to that in the application as filed or does not go beyond the application as filed, as appropriate, were furnished.

4. Additional comments:

Box No. V Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Yes:	Claims	1-20
	No:	Claims	
Inventive step (IS)	Yes:	Claims	1-20
	No:	Claims	
Industrial applicability (IA)	Yes:	Claims	1-20
	No:	Claims	

2. Citations and explanations

see separate sheet

Re Item V.

1 The following documents are referred to in this communication:
D1 : US 2003/164830 A1 (KENT OSMAN) 4 September 2003 (2003-09-04)
D2: US-B1-6 417 858 (BOSCH DEREK ET AL) 9 July 2002 (2002-07-09)
D3: US-B1-6 353 439 (LINDHOLM JOHN ERIK ET AL) 5 March 2002 (2002-03-05)
D4 : BRETERNITZ M ET AL: "Compilation, architectural support,and evaluation of SIMD graphics pipeline programs on a general-purpose CPU" 27 September 2003 (2003-09-27), PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 2003. PACT 2003. PROCEEDINGS. 12TH INTERNATIONAL CONFERENCE ON 27 SEPT. - 1 OCT. 2003, PISCATAWAY, NJ, USA,IEEE, PAGE(S) 135-145 , XP010662182 ISBN: 0-7695-2021-9

2 Document D1, which is considered to represent the most relevant state of the art for the subject-matter of claim 1, discloses (the references in parentheses applying to this document) a graphics processor comprising a shader ("Shading Unit", see paragraph 79) connected to a "Pixel Unit" by a private data path. A "Vertex Shading Unit" performs the vertex operations on the vertices entered in double buffered input buffers in round robin fashion.
An arbiter (in the "Context Unit", see paragraph 102) selects one of a plurality of

input screensa

n d d y n a m i c a l l y s w i t c h e s b e t w e e n t h e m .

Arbitration is offered in the event of a protest

a t i n g b e t w e e n t h e t e x t u r e p i p e s a t t h e l e

v e l o f t h e i e x t u r e u n i t s , t h e r e s u l t o f w h

ischi spassse docon tcoth eshaa deru nit(p arag

r
a
p
h
s
1
5
4
-
1
5
6
)

From this, the subject-matter of independent claim 1 differs in that the shader performs both vertex operations and pixel operations (performing one of the vertex operations or pixel operations based on a selected input), thus constituting a "unified shader" in the sense of the application and providing an appearance attribute.

2.1 The subject-matter of claim 1 is therefore novel (Article 33(2) PCT)

3 Document D2, which is considered to represent the most relevant state of the art for the subject-matter of claim 14, discloses (the references in parentheses applying to this document):

D2 discloses a sequencer ("main sequencer" 515) controlling instructions for inter alia the shader unit (360).

A similar system is disclosed in D3.

From this, the subject-matter of independent claim 1 differs in that the sequencer is in a unified shader in the sense of the application.

3.1 The subject-matter of claim 14 is therefore novel (Article 33(2) PCT)

4 The problem to be solved by the present invention may be regarded as to design a shader able to simultaneously perform vertex manipulations and pixel manipulations at various degrees of completion and to freely and quickly switch between the program instructions for performing such operations.

4.1 The solution to this problem proposed in claims 1 and 14 of the present application is considered as involving an inventive step (Article 33(3) PCT) because the available prior art teaches away from a unified shader performing vertex operations and pixel operations (performing one of the vertex operations or pixel operations based on a selected input) as claimed in claims 1 and 14, since the prior art uses the vertex shader and the pixel shader in different phases of a graphics operation algorithms (see D4 page 2, left-hand column, paragraph 4 - right-hand column, paragraph 3) and locates them in different entities (see D4 page 2 right-hand column last paragraph - page 3, left-hand column, first paragraph).

5 Although claims 1 and 14 have been drafted as separate independent claims, they appear to relate effectively to the same subject-matter (unified shader) and to differ from each other only with regard to the definition of the subject-matter for which protection is sought. The aforementioned claims therefore lack conciseness and as such do not meet the requirements of Article 6 PCT.

Electronic Acknowledgement Receipt

EFS ID:	10516788
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp/Christine Wright
Filer Authorized By:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001
Receipt Date:	14-JUL-2011
Filing Date:	17-MAY-2011
Time Stamp:	10:53:09
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Transmittal Letter	360001_IDSCoverSheet.pdf	18220 9d16005cb579bb43c005776d9c540cb52221b70f	no	1

Warnings:	
Information:	

TCL 1002

2	Information Disclosure Statement (IDS) Form (SB08)	360001_IDS.pdf	614273 63418bb63ea68e1617a9a01ae9c60a8f6d73388c	no	8
Warnings:					
Information:					
3	Foreign Reference	EP2299408A2.pdf	246983 09f1ee596dd08ae098b8de90753cca71578fe874	no	10
Warnings:					
Information:					
4	Foreign Reference	EP2309460A1.pdf	328840 652151662ba7749073d0478aad6f3179399113ab	no	13
Warnings:					
Information:					
5	Foreign Reference	EP2296116A2.pdf	251856 3837df67af513987e35c30ecd59d833d5cc5767e	no	10
Warnings:					
Information:					
6	Non Patent Literature	NPL1.pdf	97607 48c81a430e79648900c23d28f2748cc808f3c064	no	3
Warnings:					
Information:					
7	Non Patent Literature	NPL2.pdf	1165150 a4a9767027a73f7b846165895994d40515147e9d	no	12
Warnings:					
Information:					
8	Non Patent Literature	NPL3.pdf	862733 d8b6fc637196ac63024fdcad91205f2d4fa38e63	no	10
Warnings:					
Information:					
9	Non Patent Literature	NPL4.pdf	1641023 9a70a0366ccb14a76cb08fa9239cfd5d4ba00046	no	11
Warnings:					
Information:					
10	Non Patent Literature	NPL5.pdf	795051 9e7a348ae054c810318a669211fb8a8479fcf6e2	no	15
Warnings:					
Information:					

TCL 1002

11	Non Patent Literature	NPL6.pdf	128778 5e72407f302a6fc732392e92519bdcef4a43e48e	no	3
Warnings:					
Information:					
12	Non Patent Literature	NPL7.pdf	85580 c80ba23171009ea917b5af1b38bb401073d4334d	no	2
Warnings:					
Information:					
13	Non Patent Literature	NPL8.pdf	125523 3015062950315b0fc5eee6fb58c9ef3457cedce5	no	3
Warnings:					
Information:					
14	Non Patent Literature	NPL9.pdf	94932 f03209cd7e559d624e99a1ca9eda5062b82dd118	no	2
Warnings:					
Information:					
15	Non Patent Literature	NPL10.pdf	1320776 08d008e350c1d0391f8d16ea605cdb10b73ca4ea	no	12
Warnings:					
Information:					
16	Non Patent Literature	NPL12.pdf	27430 934b6429d64be5f3326e40de2a1fefc12a210043	no	1
Warnings:					
Information:					
17	Non Patent Literature	NPL13.pdf	95853 b97d9c1be4736c1e004665027a21f369d44eb22e	no	2
Warnings:					
Information:					
18	Non Patent Literature	NPL14.pdf	791431 e7424e47ecbdf946ad92946192b1b43afa31b54b	no	13
Warnings:					
Information:					
19	Non Patent Literature	NPL11.pdf	1258222 9b87dd5c1e49d6956277a8777615f14b72adbbe4	no	10
Warnings:					
Information:					

Total Files Size (in bytes):	9950261

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Steven Morein et al. Examiner: na
Serial No.: 13/109,738 Art Unit: na
Filing Date: May 17, 2011 Docket No.: 00100.36.0001
Confirmation No.: 2020
Title: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

INFORMATION DISCLOSURE STATEMENT IN ACCORDANCE WITH 37 CFR §§ 1.97(b) AND 1.98

Pursuant to 37 CFR §§ 1.97(b)(3) and 1.98, Applicants respectfully submit Form PTO/SB/08A. The submission of the listed document is not an admission that the information is prior art, analogous or otherwise material. It is respectfully requested that the listed document be considered and made of record in the present application.

Respectfully submitted,

Date: July 14, 2011

By: /Christopher J. Reckamp/
Christopher J. Reckamp
Registration No. 34,414

Vedder Price P.C.
222 N. LaSalle Street
Chicago, IL 60601
(312) 609-7500
FAX: (312) 609-5005

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

29153 7590 07/21/2011

ADVANCED MICRO DEVICES, INC.
C/O VEDDER PRICE P.C.
222 N.LASALLE STREET
CHICAGO, IL 60601

EXAMINER	
WASHBURN, DANIEL C	

ART UNIT	PAPER NUMBER
2628	

MAIL DATE	DELIVERY MODE
07/21/2011	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	13/109,738	MOREIN ET AL.	
	Examiner	Art Unit	
	DANIEL WASHBURN	2628	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 May 2011*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-16* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-16* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *17 May 2011* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-16 are rejected under 35 U.S.C. 102(e) as being anticipated by Lindholm (US 7,038,685).

RE claim 1, Lindholm describes a method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data (

3:59-65: “Programmable Graphics Processing Pipeline 150 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample or any other data. For simplicity, the remainder of this description will use the term 'samples' to refer to graphics data such as surfaces, primitives, vertices, pixels, fragments, or the like."

6:38-59: “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

7:6-10: “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities".

7:36-43: “Once a thread is assigned to a source sample, the thread is allocated storage resources such as locations in a Register File 350 to retain intermediate data generated during execution of program instructions associated with the thread."

9:33-56: "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

15:7-13: “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, Lindholm describes performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block (sample data, such as vertex or pixel data, is transmitted to Register File 350) and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data (the multi-threaded processing unit 400 carries out vertex operations on vertex data unless the Register File 350 doesn’t have enough room to store the incoming vertex data, in which case the thread associated with the vertex data and vertex operations must wait until enough space becomes available); and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available (

7:6-21: “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

8:15-58: "Thread Selection Unit 415 reads one or more thread entries based on thread execution priorities and outputs selected thread entries to Instruction Cache 410. Instruction cache 410 determines if the program instructions corresponding to the program counters and sample type included in the thread state data for each thread entry are available in Instruction Cache 410 ... The program instructions corresponding to the program counters from the one or more thread entries are output by Instruction Cache 410 to … Instruction Scheduler 430 … Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU [instruction window unit] 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350. An instruction specifies the location of source data needed to execute the instruction."

15:7-13: “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, Lindholm is considered to describe an embodiment including continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available, as the Execution Unit 470 may be carrying out calculations for one or more high priority pixel threads based on instructions stored in Instruction Cache 410 and/or IWU 435 while a low priority vertex thread is waiting for the one or more pixel threads to finish such that when the pixel threads finish the system will deallocate the resources assigned to the completed pixel threads in the Register File 350 and will allocate the requested amount of resources to the queued up vertex thread).

RE claim 2, Lindholm describes a unified shader, comprising:

a general purpose register block for maintaining data (

7:37-43: “Once a thread is assigned to a source sample, the thread is allocated storage resources such as locations in a Register File 350 to retain intermediate data generated during execution of program instructions associated with the thread.”);

a processor unit (FIG. 4 “Execution Unit 470” and “PCU 375”);

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block (

8:33-9:32 “Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350."

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."); and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs (

9:39-46 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … and output the processed sample to a destination specified by the instruction. The destination may be Vertex Output Buffer 260, Pixel Output Buffer 270, or Register File 350.”

4:42-5:35 “Execution Pipelines 240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device

coordinates … Execution Pipelines 240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 260 ... Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted ... programmable computation units (PCUs) within an Execution Pipeline 240 ... perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like. Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270."

Thus, the Execution Unit 470 is considered a processor unit that executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs (also see 4:22-5:35)).

RE claim 3, Lindholm describes a unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations (FIG. 4 “Execution Unit 470” and “PCU 375”.

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

Thus, the Execution Unit 470 and internal PCU 375 are collectively considered a processor unit operative to perform vertex calculation operations and pixel calculation operations); and

shared resources, operatively coupled to the processor unit (FIG. 4 illustrates Register File 350 coupled to Execution Unit 470, and 7:37-43 describes that the Register File 350 is shared among threads);

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations (7:37-43, all types of processing threads can use the Register File 350, where thread types include vertex and pixel threads (see 6:43-44).

7:6-36 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, when pixel threads have priority over vertex threads the processor unit will allocate the pixel data to the Register File 350 and will perform pixel calculation operations until enough shared resources become available in the Register File 350 to begin carrying out vertex threads, which may happen as a result of a completion of most of the pixel threads or a shift in priority such that the vertex threads now have the highest priority, and then use the Register File 350 to perform vertex calculation operations.

RE claim 4, Lindholm describes a unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations (see the corresponding section in the rejection of claim 3); and

shared resources, operatively coupled to the processor unit (see the corresponding section in the rejection of claim 3);

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations (7:37-43, all types of processing threads can use the Register File 350, where thread types include vertex and pixel threads (see 6:43-44).

7:6-36 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, when vertex threads have priority over pixel threads the processor unit will allocate the vertex data to the Register File 350 and will perform vertex calculation operations until enough shared resources become available in the Register File 350 to begin carrying out pixel threads, which may happen as a result of a completion of most of the vertex threads or a shift in priority such that the pixel threads now have the highest priority, and then use the Register File 350 to perform pixel calculation operations.

RE claim 5, Lindholm describes a unified shader comprising:

a processor unit (FIG. 4 “Execution Unit 470” and “PCU 375”);

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store (

8:33-9:32 “Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350."

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

7:6-10 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities".

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, the Scheduler 430 and Instruction Dispatcher 440 are collectively considered a sequencer coupled to the Execution Unit 470, the sequencer maintaining instructions operative to cause the Execution Unit 470 to execute vertex calculation and pixel calculation operations on selected data maintained in a Register File 350 depending upon an amount of space available in the Register File 350).

RE claim 6, Lindholm describes the shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory (

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350").

RE claim 7, Lindholm describes the shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal (

6:60-7:36 "Thread allocation priority, as described further herein, is used to assign a thread to a source sample. A thread allocation priority is specified for each sample type and Thread Control Unit 420 is configured to assign threads to samples or allocate locations in a Register File 350 based on the priority assigned to each sample type. The thread allocation priority may be fixed, programmable, or dynamic."

The Thread Control Unit 420 is considered a selection circuit operative to provide information to the store (Register File 350) in response to a control signal, where the control signal is the thread allocation priority associated with each thread or thread type).

RE claim 8, Lindholm describes the shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs (

5:11-35 "Pixel Input Buffer 215 outputs the samples to each Execution Pipeline 240 … Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted ... programmable computation units (PCUs) within an Execution Pipeline 240 ... perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like. Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270.").

RE claim 9, Lindholm describes the shader of claim 5, wherein the processor unit executes vertex calculations while the pixel calculations are still in progress (

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:39-49 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”).

RE claim 10, Lindholm describes the shader of claim 5, wherein the processor

unit generates vertex position and appearance data in response to a selected one of the

plurality of inputs (

4:42-5:35 “Execution Pipelines 240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates … Execution Pipelines 240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 260”).

RE claim 11, Lindholm describes the shader of claim 7, wherein the control

signal is provided by an arbiter (

6:60-7:36 “Thread allocation priority, as described further herein, is used to assign a thread to a source sample. A thread allocation priority is specified for each sample type and Thread Control Unit 420 is configured to assign threads to samples or allocate locations in a Register File 350 based on the priority assigned to each sample type. The thread allocation priority may be fixed, programmable, or dynamic … In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220 ... In a further alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on graphics primitive size”.

Thus, while an arbiter isn't explicitly described, the Examiner considers it inherent that some portion of the system acts as an arbiter, and therefore can be considered an arbiter, as some portion of the system assigns priorities to thread and sample types according to the current processing circumstances, in order to more efficiently process the data).

RE claim 12, Lindholm describes a graphics processor comprising:

a unified shader comprising a processor unit that executes vertex calculations while the pixel calculations are still in progress (

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:39-49 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”).

RE claim 13, Lindholm describes the graphics processor of claim 12 wherein the unified shader comprises a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store (see the corresponding section in the rejection of claim 5).

RE claim 14, Lindholm describes the graphics processor of claim 12 comprising a vertex block operative to fetch vertex information from memory (see the rejection of claim 6).

RE claim 15, Lindholm describes a unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload (

7:6-36 “Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220 … In a further alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on graphics primitive size (number of pixels or fragments included in a primitive)”.

9:39-49 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”).

RE claim 16, Lindholm describes the shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store (FIG. 4 and 8:15-46 describes Instruction Cache 410, which is considered an instruction store.

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations… Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”

Thus, the Execution Unit 470 performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store).

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DANIEL WASHBURN whose telephone number is (571)272-5551. The examiner can normally be reached on 9:30 A.M. to 6 P.M..

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ulka Chauhan can be reached on 571-272-7782. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/DANIEL WASHBURN/
Primary Examiner, Art Unit 2628
7/12/11

Notice of References Cited	Application/Control No. 13/109,738	Applicant(s)/Patent Under Reexamination MOREIN ET AL.	
	Examiner DANIEL WASHBURN	Art Unit 2628	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-7,038,685	05-2006	Lindholm, John Erik	345/501
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	DANIEL WASHBURN	2628

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	07/12/2011								
	1	✓								
	2	✓								
	3	✓								
	4	✓								
	5	✓								
	6	✓								
	7	✓								
	8	✓								
	9	✓								
	10	✓								
	11	✓								
	12	✓								
	13	✓								
	14	✓								
	15	✓								
	16	✓								

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	DANIEL WASHBURN	2628

SEARCHED			
Class	Subclass	Date	Examiner
345	501	7/12/11	DW

SEARCH NOTES		
Search Notes	Date	Examiner
Searched EAST (all databases) see search history printout	7/12/11	DW
Also see search histories for apps 12/791,597 and 11/842,256	7/12/11	DW
conducted inventor name search	7/12/11	DW

INTERFERENCE SEARCH			
Class	Subclass	Date	Examiner

	/DANIEL WASHBURN/ Primary Examiner.Art Unit 2628

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 2020

SERIAL NUMBER	FILING or 371(c) DATE RULE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
13/109,738	05/17/2011	345	2628	00100.36.0001

APPLICANTS

Stephen Morein, Cambridge, MA;
Laurent Lefebvre, Lachgnaie, CANADA;
Andy Gruber, Arlington, MA;
Andi Skende, Shrewsbury, MA;

**** CONTINUING DATA ***************************

This application is a CON of 12/791,597 06/01/2010 ABN
which is a CON of 11/842,256 08/21/2007 ABN
which is a CON of 11/117,863 04/29/2005 PAT 7,327,369
which is a CON of 10/718,318 11/20/2003 PAT 6,897,871

**** FOREIGN APPLICATIONS ***************************

**** IF REQUIRED, FOREIGN FILING LICENSE GRANTED ****

05/27/2011

Foreign Priority claimed	☐ Yes ☑ No	☐ Met after Allowance	STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS	INDEPENDENT CLAIMS
35 USC 119(a-d) conditions met	☐ Yes ☑ No		MA	5	16	7
Verified and Acknowledged	/DANIEL C WASHBURN/ Examiner's Signature	Initials				

ADDRESS

ADVANCED MICRO DEVICES, INC.
C/O VEDDER PRICE P.C.
222 N.LASALLE STREET
CHICAGO, IL 60601
UNITED STATES

TITLE

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

FILING FEE RECEIVED		
1970	FEES: Authority has been given in Paper No._____ to charge/credit DEPOSIT ACCOUNT No._____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	1	("7038685").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2011/07/12 13:27
L2	1217	345/501.ccls.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2011/07/12 13:29

EAST Search History (Interference)

<This search history is empty>

7/12/2011 1:53:40 PM
C:\Documents and Settings\dwashburn1\My Documents\EAST\Workspaces\Morein et al. 11117863.wsp

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
13/109,738	05/17/2011	Stephen Morein	00100.36.0001

CONFIRMATION NO. 2020

PUBLICATION NOTICE

29153
ADVANCED MICRO DEVICES, INC.
C/O VEDDER PRICE P.C.
222 N.LASALLE STREET
CHICAGO, IL 60601

OC000000049722760

Title:GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

Publication No.US-2011-0216077-A1
Publication Date:09/08/2011

NOTICE OF PUBLICATION OF APPLICATION

The above-identified application will be electronically published as a patent application publication pursuant to 37 CFR 1.211, et seq. The patent application publication number and publication date are set forth above.

The publication may be accessed through the USPTO's publically available Searchable Databases via the Internet at www.uspto.gov. The direct link to access the publication is currently http://www.uspto.gov/patft/.

The publication process established by the Office does not provide for mailing a copy of the publication to applicant. A copy of the publication may be obtained from the Office upon payment of the appropriate fee set forth in 37 CFR 1.19(a)(1). Orders for copies of patent application publications are handled by the USPTO's Office of Public Records. The Office of Public Records can be reached by telephone at (703) 308-9726 or (800) 972-6382, by facsimile at (703) 305-8759, by mail addressed to the United States Patent and Trademark Office, Office of Public Records, Alexandria, VA 22313-1450 or via the Internet.

In addition, information on the status of the application, including the mailing date of Office actions and the dates of receipt of correspondence filed in the Office, may also be accessed via the Internet through the Patent Electronic Business Center at www.uspto.gov using the public side of the Patent Application Information and Retrieval (PAIR) system. The direct link to access this status information is currently http://pair.uspto.gov/. Prior to publication, such status information is confidential and may only be obtained by applicant using the private side of PAIR.

Further assistance in electronically accessing the publication, or about PAIR, is available by calling the Patent Electronic Business Center at 1-866-217-9197.

Office of Data Managment, Application Assistance Unit (571) 272-4000, or (571) 272-4200, or 1-888-786-0101

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Stephen Morein et al.	Examiner:	Daniel C. Washburn
Serial No.:	13/109,738	Art Unit:	2628
Filing Date:	May 17, 2011	Docket No.:	00100.36.0001

Confirmation No.: 2020

Title: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

RESPONSE

Dear Sir:

In response to the office action mailed July 21, 2011, Applicants petition for a three month extension of time and respond as follows:

Listing of the Claims begins on page 2 of this paper.

Remarks begin on page 6 of this paper.

Listing of the Claims:

1. (original) A method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

2. (original) A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.

3. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

4. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

5. (original) A unified shader comprising:

a processor unit;

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

6. (original) The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.

7. (original) The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.

8. (original) The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs.

9. (original) The shader of claim 5, wherein the processor unit executes vertex calculations while the pixel calculations are still in progress.

10. (original) The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.

11. (original) The shader of claim 7, wherein the control signal is provided by an arbiter.

12. (original) A graphics processor comprising:

a unified shader comprising a processor unit that executes vertex calculations while the pixel calculations are still in progress.

13. (original) The graphics processor of claim 12 wherein the unified shader comprises a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

14. (original) The graphics processor of claim 12 comprising a vertex block operative to fetch vertex information from memory.

15. (original) A unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.

16. (original) The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

REMARKS

Applicants respectfully traverse and request reconsideration.

Claims 1-16 stand rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by U.S. Patent No. 7,038,685 (Lindholm). Applicants respectfully submit herewith Declarations under 37 C.F.R. § 1.131 for inventors Laurent Lefebvre, Andrew E. Gruber, Stephen L. Morein and Andi P. Skende establishing conception and reduction to practice of the currently claimed subject matter prior to the June 30, 2003 priority date of Lindholm. It is believed that Lindholm does not claim the same patentable invention as defined by 37 C.F.R. § 41.203(a) and that the present rejection is not based on a statutory bar, i.e., Lindholm qualifies as prior art only under 35 U.S.C. § 102(e). Accordingly, the attached Declarations are relied on to establish prior reduction to practice of the claimed subject matter, particularly with regard to independent claims 1-5, 12 and 15.

Regarding the reduction to practice evidenced by the attached Declarations, Applicants first note that, properly presented, a Rule 131 declaration may demonstrate prior invention if it provides a "showing of facts . . . as to establish reduction to practice prior to the effective date of the reference." 37 C.F.R. § 1.131(b). As set forth in M.P.E.P. § 715.07(I), evidence in support of asserted facts demonstrating prior invention may be provided in the form of "an accompanying model." With regard to an apparatus and/or process implemented by an integrated circuit or the like, Applicants respectfully submit that a simulation of such an apparatus and/or circuit may effectively serve as a "model" demonstrating successful reduction to practice. Specifically, Applicants respectfully submit that evidence of (i) a successful computer-based simulation of a physical embodiment and/or (ii) a description of a physical embodiment capable of translation to implement the actual physical embodiment, coupled with

successfully testing of the resulting physical embodiment is sufficient to demonstrate an actual reduction to practice for the purposes of Rule 131 declaration. (See *McDonnell Douglas Corp. v. U.S.*, 670 F. 2d 156, 161 (Ct. Cl. 1982) (where court concludes that "physical tests proved that the computer approved device . . . failed in actual practice . . . to perform in the manner intended" and that subsequent successful physical testing was the first reduction to practice, a necessary implication is that a valid reduction to practice would result if actual physical testing demonstrates that prior computer simulation was adequate.); *Mosaid Tech. Inc, v. Samsung Elec. Co.*, 362 F.Supp.2d 526, 547 (D.N.J. 2005) (noting that the *McDonnell* case suggested "that a computer simulation may be a valid reduction to practice, but not if subsequent, actual physical testing proves that it is inadequate," and that "there are areas of science where a successfully run simulation represents the end of the inventive process and the construction of the physical embodiment is but a matter of mere routine and mechanical application [such that] a simulation should be a valid reduction to practice."))

With regard to the instant application, as shown in the attached Declarations, Applicants have provided evidence that both a simulation and hardware design description (expressed in a hardware design language capable of conversion to a physical embodiment) subsequently lead to a successfully tested physical embodiment of (and, therefore, actual reduction to practice of) the subject matter recited in the independent claims. More particularly, the attached Declarations demonstrate invention of the recited subject matter in claims 1-5, 12 and 15 prior to the effective filing date of the Lindholm reference.

Thus, in light of the Declarations, Applicants respectfully submit that Lindholm is not available as prior art against, and therefore obviates the sole basis for rejecting, the above claims, which claims are therefore in suitable condition for allowance. Applicants further note that

claims 6-11, 13, 14 and 16 are dependent upon, and therefore incorporate the limitations of, respective ones of claims 5, 12 and 15. As such, claims 6-11, 13, 14 and 16 are also allowable for the same reasons presented above relative to their respective independent claims.

Applicant respectfully submits that the claims are now believed to be in condition for allowance and that a timely Notice of Allowance be issued in this case. If the Examiner believes that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at (312) 356-5094.

Respectfully submitted,

Dated: January 18, 2012

By: /Christopher J. Reckamp/
Christopher J. Reckamp
Reg. No. 34,414

Faegre Baker Daniels LLP
311 S. Wacker Drive
Chicago, IL 60606
PHONE: (312) 356-5094
FAX: (312) 212-6501

PATENT
ATTORNEY DOCKET NO. 00100.36.0001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Stephen Morein et al.	)
Serial No.	13/109,738	) Art Unit: 2628
Filed:	May 17, 2011	) Examiner: Daniel C. Washburn
For: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER		) Confirmation No. 2020

DECLARATION UNDER 37 C.F.R. § 1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22314-1450

Sir:

We, Stephen L Morein a citizen of the U.S. residing at 367 Santana Heights, Unit 3027, San Jose California 95128; Laurent Lefebvre, a citizen of Canada residing at 1055 Garden Avenue, Mascouche, Quebec, CANADA, J7L-0A1; Andrew E. Gruber a citizen of the U.S. residing at 251 Pleasant Street, Arlington, Massachusetts 02476; and Andi Petrit Skende a citizen of the U.S. residing at 35 Sunrise Avenue, Worcester, Massachusetts 01606, do hereby declare that:

1. We are joint inventors of the subject matter described and claimed in U.S. Patent Application No. 13/109,738 (hereinafter "the Invention"), filed in the United States of America on May 17, 2011, entitled "Graphics Processing Architecture Employing a Unified Shader", which application is a continuation of and claims priority to U.S. Patent Application No. 12/791,597 filed June 1, 2010, which application is a continuation of and claims priority to U.S. Patent Application No. 11/842,256 filed August 21, 2007, which application is a continuation of and claims priority to U.S. Patent Application No. 11/117,863 filed April 29, 2005 (now U.S. Patent No. 7,327,369), which application is a continuation of and claims priority to U.S. Patent Application No. 10/718,318 filed November 20, 2003 (hereinafter "the '318 application"; now U.S. Patent No. 6,897,871).

2. We conceived the Invention prior to June 30, 2003 while employed by ATI Technologies Inc. and/or one of its wholly owned subsidiaries ("ATI") as indicated by attached Exhibits A and B. Exhibit A is a copy of emulation code files entitled Reg_file.cpp, Instruction_store.cpp, Arbiter.cpp, Gpr_manager.cpp, sq_alu.cpp and sq_block_model.cpp that,

based on information and belief, we invented and assisted in coding prior to June 30, 2003 the ("Model Code"). Exhibit B is a copy of hardware register transfer level (RTL) files ("the Chip Design Code") entitled sq_gpr_alloc.v, Sq_alu_instr_seq.v, sq_instruction_store_v, sp_macc_gpr.v, sp_vector.v, sq.v, sq_export_alloc.v, sq_ctl_flow_seq.v, sq_alu_instr_seq.v, sq_thread_arb.v and sq_shader_seq.v, that, based on information and belief, we assisted in creating prior to June 30, 2003. Prior to June 30, 2003 we created a graphics processing system that operated as claimed using a computer system that successfully executed the Model Code. Prior to June 30, 2003 we also created a graphics processing system as claimed in the form of a computer system that used an RTL simulator to successfully validate the operation of an integrated circuit version of the claimed graphics processing system and method. At least the following language and citations adequately support the above:

a. As shown in Exhibit A, the Model Code comprises various software instructions written in the well-known C++ language. When executed by the computer system , the Model Code caused the computer system to operate as claimed in at least claims 1-5, 12 and 15 of the Invention.

b. Using the Model Code, we successfully verified the operation of the claimed subject matter for its intended purpose through emulation thereof.

c. As shown in Exhibit B, the Chip Design Code comprises various instructions written in a well-known hardware description language. The Chip Design Code was used by an RTL simulator system to validate the operation of an integrated circuit version of the claimed graphics processing system and method as claimed in at least claims 1-5, 12 and 15. As further known by practitioners in the field of integrated circuit design, such instructions are used to generate gate level detail for silicon fabrication.

d. On information and belief, the computer system operating the Model Code and the RTL simulator system operating the Chip Design Code represents the claimed structure and operation embodied in an integrated graphics processing circuit chip referred to as the ATI XENOS chip produced by ATI on or about October, 2004 that was incorporated in the XBOX 360 product.

Accordingly, the contents of Exhibits A and B establish the possession by us of the whole Invention, falling within the scope of currently pending claims, such as but not limited to at least claims 1-5, 12 and 15.

3. Each of us hereby declare that all statements made herein are of my own knowledge, are true and that all statements made on information and belief are believed to be true; and each of us further declare that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued therefrom.

Dated: NOV 24, 2011

Stephen L. Morein

Dated: _____

Laurent Lefebvre

Dated: _____

Andrew E. Gruber

Dated: _____

Andi Petrit Skende

3. Each of us hereby declare that all statements made herein are of my own knowledge, are true and that all statements made on information and belief are believed to be true; and each of us further declare that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued therefrom.

Dated:_____

Stephen L. Morein

Dated: OCTOBER 4, 2011

Laurent Lefebvre

Dated:_____

Andrew E. Gruber

Dated:_____

Andi Petrit Skende

3. Each of us hereby declare that all statements made herein are of my own knowledge, are true and that all statements made on information and belief are believed to be true; and each of us further declare that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued therefrom.

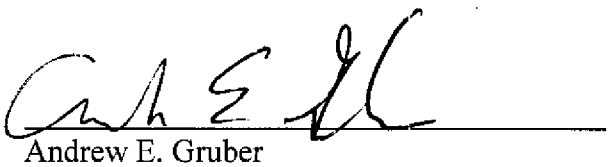
Dated:_____

_____
Stephen L. Morein

Dated:_____

_____
Laurent Lefebvre

Dated: 10/23/11


Andrew E. Gruber

Dated:_____

_____
Andi Petrit Skende

3. Each of us hereby declare that all statements made herein are of my own knowledge, are true and that all statements made on information and belief are believed to be true; and each of us further declare that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued therefrom.

Dated:_____

Stephen L. Morein

Dated:_____

Laurent Lefebvre

Dated:_____

Andrew E. Gruber

Dated: December 22, 2011

Andi Petrit Skende

Reg_file.cpp

```
#include "reg_file.h"


RegFile::RegFile()
{
	for (int i=0;i<128;i++)
		for (int j=0;j<16;j++)
			for (int k=0;k<4;k++)
				regValues[i].Val[j].field[k].clamp(0);
}

void RegFile::GetConstValues(const RegVect* &Values,int Addr)
{
	Values = &(regValues[Addr].Val[0]);
}

void RegFile::GetValues(RegVect* &Values,int Addr)
{
	Values = &(regValues[Addr].Val[0]);
}
```

Instruction store:

Instruction_store.cpp

```
#include "instruction_store.h"


IStore::IStore()
{
	for (int i=0;i<4096;i++)
	{
		instructions[i].byte0=0x00;
		instructions[i].byte1=0x00;
		instructions[i].byte2=0x00;
		instructions[i].byte3=0x00;
		instructions[i].byte4=0x00;
		instructions[i].byte5=0x00;
		instructions[i].byte6=0x00;
		instructions[i].byte7=0x00;
		instructions[i].byte8=0x00;
		instructions[i].byte9=0x00;
		instructions[i].byte10=0x00;
		instructions[i].byte11=0x00;
	}
}

void IStore::GetInst(Instruction &inst,int addr)
{
	inst = instructions[addr];
}
```

```
void IStore::GetInst(ALU_Instruction &aluInst, int addr)
{
	aluInst.SrcASel = ((instructions[addr].byte11 & 0x80) >> 7);
	aluInst.SrcBSel = ((instructions[addr].byte11 & 0x40) >> 6);
	aluInst.SrcCSel = ((instructions[addr].byte11 & 0x20) >> 5);
	aluInst.VectorOpcode = ((instructions[addr].byte11 & 0x1F));
	aluInst.SourceARegPointer = ((instructions[addr].byte10 ));
	aluInst.SourceBRegPointer = ((instructions[addr].byte9 ));
	aluInst.SourceCRegPointer = ((instructions[addr].byte8 ));
	aluInst.Constan0RelAbs = ((instructions[addr].byte7 & 0x80) >> 7);
	aluInst.Constan1RelAbs = ((instructions[addr].byte7 & 0x40) >> 6);
	aluInst.RelativeAddrRegSel = ((instructions[addr].byte7 & 0x20) >> 5);
	aluInst.PredicateSelect = ((instructions[addr].byte7 & 0x18) >> 3);
	aluInst.SourceANegate = ((instructions[addr].byte7 & 0x04) >> 2);
	aluInst.SourceBNegate = ((instructions[addr].byte7 & 0x02) >> 1);
	aluInst.SourceCNegate = ((instructions[addr].byte7 & 0x01)     );
	aluInst.SourceASwizzle = ((instructions[addr].byte6) );
	aluInst.SourceBSwizzle = ((instructions[addr].byte5) );
	aluInst.SourceCSwizzle = ((instructions[addr].byte4) );
	aluInst.ScalarOpcode = ((instructions[addr].byte3 & 0xfc) >> 2);
	aluInst.ScalarClamp = ((instructions[addr].byte3 & 0x02) >> 1);
	aluInst.VectorClamp = ((instructions[addr].byte3 & 0x01)     );
	aluInst.ScalarWriteMask = ((instructions[addr].byte2 & 0xf0) >> 4);
	aluInst.VectorWriteMask = ((instructions[addr].byte2 & 0x0f) );
	aluInst.ScalarResultPointer = ((instructions[addr].byte1 )     );
	aluInst.VectorResultPointer = ((instructions[addr].byte0 )     );
}

void IStore::GetInst(TInstrPacked &texInst, int addr)
{
	texInst.unpack((const uint8*)(&instructions[addr]));
}

void IStore::GetInst(CF_Instruction &cfInst, int addr, bool left)
{
	// read from bytes 11 thru 6
	if (left)
	{
		cfInst.opCode = ((instructions[addr].byte11 & 0xF0) >> 4);
		cfInst.addrMode = ((instructions[addr].byte11 & 0x08) >> 3);
		cfInst.bufferSel = ((instructions[addr].byte11 & 0x06) >> 1);
		cfInst.condition = ((instructions[addr].byte11 & 0x04) >> 2);
		cfInst.boolAddr = ((instructions[addr].byte11 & 0x03) << 6) |
                                    ((instructions[addr].byte10 & 0xFC) >> 2);
		cfInst.direction = ((instructions[addr].byte10 & 0x02) >> 1);
		cfInst.instTypeSer = ((instructions[addr].byte10 & 0x03) << 16) |
                                       ((instructions[addr].byte9) << 8) |
                                          ((instructions[addr].byte8));
		cfInst.predBreak = ((instructions[addr].byte8 & 0x20) >> 5);
		cfInst.loopId = ((instructions[addr].byte8 & 0x1F));
		cfInst.count = ((instructions[addr].byte7 & 0xF0) >> 4);
		cfInst.force = ((instructions[addr].byte7 & 0x20) >> 5);
		cfInst.jcAddress = ((instructions[addr].byte7 & 0x1F) << 8) |
                                    ((instructions[addr].byte6));
		cfInst.address = ((instructions[addr].byte7 & 0x0F) << 8) |
                                    ((instructions[addr].byte6));
		cfInst.allocSize = ((instructions[addr].byte6 & 0x0F));
	}
```

```
	// read from bytes 5 thru 0
	else
	{
		cfInst.opCode = ((instructions[addr].byte5 & 0xF0) >> 4);
		cfInst.addrMode = ((instructions[addr].byte5 & 0x08) >> 3);
		cfInst.bufferSel = ((instructions[addr].byte5 & 0x06) >> 1);
		cfInst.condition = ((instructions[addr].byte5 & 0x04) >> 2);
		cfInst.boolAddr = ((instructions[addr].byte5 & 0x03) << 6) |
					((instructions[addr].byte4 & 0xFC) >> 2);
		cfInst.direction = ((instructions[addr].byte4 & 0x02) >> 1);
		cfInst.instTypeSer = ((instructions[addr].byte4 & 0x03) << 16) |
					((instructions[addr].byte3) << 8) |
					  ((instructions[addr].byte2));
		cfInst.predBreak = ((instructions[addr].byte2 & 0x20) >> 5);
		cfInst.loopId = ((instructions[addr].byte2 & 0x1F));
		cfInst.count = ((instructions[addr].byte1 & 0xF0) >> 4);
		cfInst.force = ((instructions[addr].byte1 & 0x20) >> 5);
		cfInst.jcAddress = ((instructions[addr].byte1 & 0x1F) << 8) |
					((instructions[addr].byte0));
		cfInst.address = ((instructions[addr].byte1 & 0x0F) << 8) |
					((instructions[addr].byte0));
		cfInst.allocSize = ((instructions[addr].byte0 & 0x0F));
	}
}

void IStore::SetInst(const Instruction &inst,int addr)
{
	instructions[addr]=inst;
}
```

Performing operations on pixels or vertices:

Arbiter.cpp

```
boolean Arbiter::chooseAluStation(int &lineNumber, Shader_Type &sType,
		bool otherAluRunning,const CfMachine& otherCFMachine,bool &predOn)
{
	int i;
	int vertexPick = -1;
	int pixelPick = -1;
	bool pcSpace;
	int lineCheck;
	predOn = true;

	// do pixels first
	lineCheck = pixelHead;
	for (i=0;i<pixelRsCount;i++)
	{
		if (pixelStation[lineCheck].status.valid != 0 &&
pixelStation[lineCheck].status.ressourceNeeded == ALU
		&& !pixelStation[lineCheck].status.event)
		{
			// no allocation needed
			if (pixelStation[lineCheck].status.allocation == SQ_NO_ALLOC)
			{
				pixelPick = lineCheck;
			}
			// we need to make sure there is space in the appropriate buffer
```

```
                    else if (pixelStation[lineCheck].status.allocation == SQ_MEMORY &&
(pixelStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ->GetExportBuffer()/4
                           && pendingAllocs < 2 && sq->pSX_SQ->GetValid())
                    {
                           pixelPick = lineCheck;
                    }
                    else if (pixelStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL &&
                           pixelStation[lineCheck].status.allocationSize <= sq->pSX_SQ-
>GetExportBuffer()/4
                           && pendingAllocs < 2 && sq->pSX_SQ->GetValid())
                    {
                           pixelPick = lineCheck;
                    }
                    // make sure the status says we can pick this pixel
                    if (pixelPick != -1)
                    {
                           // check for serial with texture pending
                           if (pixelStation[pixelPick].status.serial &&
                                  pixelStation[pixelPick].status.texReadsOutstanding)
                                  pixelPick = -1;
                           // if last or alloc is set we can only pick the two oldests
threads also for color exports
                           else if ((pixelStation[pixelPick].status.last
                                  || pixelStation[pixelPick].status.allocation ==
SQ_PARAMETER_PIXEL )&&
                                  !(pixelPick==pixelHead || pixelPick==((pixelHead-
1)%MAX_PIX_RESERVATION_SIZE)))
                                  pixelPick = -1;
                           // cannot pick last if texture reads are outstanding
                           else if (pixelStation[pixelPick].status.last &&
                                  pixelStation[pixelPick].status.texReadsOutstanding)
                                  pixelPick = -1;
                           // can only pick the second to old if the first is already
running and last is set
                           else if (pixelStation[pixelPick].status.last && pixelHead !=
pixelPick)
                           {
                                  if (pixelStation[pixelPick].status.first ||
!pixelStation[pixelHead].status.last
                                         || pixelStation[pixelHead].status.valid)
                                         pixelPick = -1;
                                  else
                                  {
                                         predOn = false;
                                         break;
                                  }
                           }
                           else
                                  break;
                    }
             }// endif pixels

             lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
      }// end for loop

      lineCheck = vertexHead;
      for (i=0;i<vertexRsCount;i++)
```

```
	{
		if (vertexStation[lineCheck].status.valid != 0 &&
vertexStation[lineCheck].status.ressourceNeeded == ALU
			&&!vertexStation[lineCheck].status.event)
		{
			// no allocation needed
			if (vertexStation[lineCheck].status.allocation == SQ_NO_ALLOC)
			{
				vertexPick = lineCheck;
			}
			// we need to make sure there is space in the appropriate buffer
			else
			{
				if (vertexStation[lineCheck].status.allocation == SQ_MEMORY)
				{
					if
(((vertexStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ-
>GetExportBuffer()/4)
						&& sq->pSX_SQ->GetValid() && pendingAllocs <2)
					{
						vertexPick = lineCheck;
					}
				}
				else if (vertexStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL)
				{
					// determine if there is space in the PCs for an
eventual PC export
					pcSpace =
checkPC((vertexStation[lineCheck].status.allocationSize+1)*4);
					if (pcSpace)
					{
						// make sure every older threads have their
position allocated
						bool alloc_done = true;
						int alloc_line = vertexHead;
						while (lineCheck != alloc_line)
						{
							if
(vertexStation[alloc_line].status.pcAllocated == false)
							{
								alloc_done = false;
								break;
							}
							alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
						}
						if (alloc_done)
						{
							vertexPick = lineCheck;
						}
					}
				}
				else if (vertexStation[lineCheck].status.allocation ==
SQ_POSITION
					&& (sq->pSX_SQ->GetPositionReady() >=
vertexStation[lineCheck].status.allocationSize )
					&& sq->pSX_SQ->GetValid()
```

```
                                && pendingAllocs <2)
                        {
                                // make sure every older threads have their position
allocated
                                bool alloc_done = true;
                                int alloc_line = vertexHead;
                                while (lineCheck != alloc_line)
                                {
                                        if
(vertexStation[alloc_line].status.posAllocated == false)
                                        {
                                                alloc_done = false;
                                                break;
                                        }
                                        alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                }
                                if (alloc_done)
                                {
                                        vertexPick = lineCheck;
                                }
                        }
                }
                // make sure the status says we can pick this vertex
                if (vertexPick != -1)
                {
                        // check for serial with texture pending
                        if (vertexStation[vertexPick].status.serial &&
                                vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        // if last is set we can only pick the two oldests threads
                        else if (vertexStation[vertexPick].status.last &&
                                !(vertexPick==vertexHead || vertexPick==((vertexHead-
1)%MAX_VTX_RESERVATION_SIZE)))
                                vertexPick = -1;
                        // cannot pick last if texture reads are outstanding
                        else if (vertexStation[vertexPick].status.last &&
                                vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        // can only pick the second to old if the first is already
running
                        else if ((vertexStation[vertexPick].status.last) && vertexHead
!= vertexPick)
                        {
                                if (vertexStation[vertexPick].status.first ||
!vertexStation[vertexHead].status.last
                                        || vertexStation[vertexHead].status.valid)
                                        vertexPick = -1;
                                else
                                {
                                        predOn = false;
                                        break;
                                }
                        }
                        else
                                break;
                }
        }// endif vertex
```

```
		lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
	}// end for loop

	// right now vertices have priority over pixels always,
	// will have to change this when the registers are there.
	if (vertexPick != -1)
	{
		lineNumber = vertexPick;
		sType = VERTEX;

		// HERE WE MUST DO THE ALLOCATION
		// also send a pulse to the SX if we need a buffer (position or multipass)

		if (vertexStation[vertexPick].status.allocation != SQ_NO_ALLOC)
		{
			// parameter cache allocation
			if (vertexStation[vertexPick].status.allocation ==
SQ_PARAMETER_PIXEL)
			{
				vertexStation[vertexPick].status.pcAllocated = true;
				vertexStation[vertexPick].data.pcBasePtr = sq->pcHead;
				vertexStation[vertexPick].data.exportId = 0;

				if (sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4 < 128)
				{
					sq->pcHead = sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4;
				}
				else
				{
					sq->pcHead =
(vertexStation[vertexPick].status.allocationSize)*4-(128-sq->pcHead);
					sq->checkHigh = !sq->checkHigh;
				}
				sq-
>pcAllocated.push((vertexStation[vertexPick].status.allocationSize)*4);
			}
			// position
			else if (vertexStation[vertexPick].status.allocation == SQ_POSITION)
			{
				// starting a new allocation
				pendingAllocs ++;

				vertexStation[vertexPick].status.posAllocated = true;
				vertexStation[vertexPick].status.pulseSx = true;
				sq->pSQ_SX->SetValid(true);
				uinteger<3> st;
				st = vertexStation[vertexPick].data.state;
				sq->pSQ_SX->SetSQ_SX_exp_state(st);
				sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
				vertexStation[vertexPick].data.exportId = exportId;
				exportId = !exportId;
				uinteger<2> temp;
				temp = 2;
				sq->pSQ_SX->SetSQ_SX_exp_type(temp);
				sq->pSQ_SX->SetSQ_SX_exp_valid(true);
```

```
                temp = vertexStation[vertexPick].status.allocationSize-1;
                sq->pSQ_SX->SetSQ_SX_exp_number(temp);
        }
        // multipass
        else
        {
                // starting a new allocation
                pendingAllocs ++;

                vertexStation[vertexPick].status.pcAllocated = true;
                vertexStation[vertexPick].status.pulseSx = true;
                sq->pSQ_SX->SetValid(true);
                uinteger<3> st;
                st = vertexStation[vertexPick].data.state;
                sq->pSQ_SX->SetSQ_SX_exp_state(st);
                sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                vertexStation[vertexPick].data.exportId = exportId;
                exportId = !exportId;
                uinteger<2> temp;
                temp = 3;
                sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                temp = vertexStation[vertexPick].status.allocationSize;
                sq->pSQ_SX->SetSQ_SX_exp_number(temp);
        }

        // dump the interface
        if (sq->m_dumpSQ > 0)
        {
                sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
                if (sq->m_sqSxDump->_data.Valid)
                {
                        sq->m_sqSxDump->Dump();
                }
        }

        // clear the allocation fields
        vertexStation[vertexPick].status.allocationSize = 0;
        vertexStation[vertexPick].status.allocation = SQ_NO_ALLOC;
    }
    return true;
}
if (pixelPick != -1)
{
    lineNumber = pixelPick;
    sType = PIXEL;

    if (pixelStation[pixelPick].status.allocation != SQ_NO_ALLOC)
    {
        // starting a new allocation
        pendingAllocs ++;

        if (pixelStation[pixelPick].status.allocation == SQ_PARAMETER_PIXEL)
        {
                sq->pSQ_SX->SetValid(true);
                uinteger<3> st;
                st = pixelStation[pixelPick].data.state;
                sq->pSQ_SX->SetSQ_SX_exp_state(st);
```

```
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        pixelStation[pixelPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;

                        sq->setContextNumber(st);
                        uint8 mode = sq->SQ_PROGRAM_CNTL.getPS_EXPORT_MODE();
                        // exporting Z
                        if (mode &0x01)
                                temp = 1;
                        // not exporting Z
                        else
                                temp = 0;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = pixelStation[pixelPick].status.allocationSize-temp-1;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }
                // multipass
                else
                {
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = pixelStation[pixelPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        pixelStation[pixelPick].data.exportId = exportId;
                        pixelStation[pixelPick].status.pulseSx = true;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 3;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = pixelStation[pixelPick].status.allocationSize;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                        pixelStation[pixelPick].status.pulseSx = true;
                }

                // dump the interface
                if (sq->m_dumpSQ > 0)
                {
                        sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
                        if (sq->m_sqSxDump->_data.Valid)
                        {
                                sq->m_sqSxDump->Dump();
                        }
                }

                // clear the allocation fields
                pixelStation[pixelPick].status.allocationSize = 0;
                pixelStation[pixelPick].status.allocation = SQ_NO_ALLOC;
        }
        return true;
    }

    return false;
}
```

Checking for GPR space:
Gpr_manager.cpp

```
#include "gpr_manager.h"
#include "user_block_model.h"

GPR_manager::GPR_manager(cUSER_BLOCK_SQ *pSQ)
{
	// set the pointer to the SQ
	sq = pSQ;

	// set the limits (READ REGISTERS)
	pixLimit = sq->SQ_GPR_MANAGEMENT.REG_SIZE_PIX;
	vertLimit = 128-sq->SQ_GPR_MANAGEMENT.REG_SIZE_VTX;

	baseCountPix = 0;
	freeCountPix = 0;
	pixTestHigh = true;

	baseCountVert = 127;
	freeCountVert = 127;
	vertTestHigh = true;
}

boolean GPR_manager::testAllocate(int number_gpr,int &base_addr,Shader_Type stype)
{
	bool wrap = false;
	int testBaseCount;

	if (stype == PIXEL)
	{
		testBaseCount = baseCountPix;
		base_addr= baseCountPix;

		// special case for MAX GPRs
		if (number_gpr == pixLimit)
		{
			if (freeCountPix==baseCountPix && pixTestHigh &&
				freeCountPix != -1)
			{
				return false;
			}
			else
				return true;
		}

		if (testBaseCount + number_gpr < pixLimit)
			testBaseCount = testBaseCount + number_gpr;
		else
		{
			testBaseCount = number_gpr-(pixLimit-testBaseCount);
			// we wrapped change the test type
			pixTestHigh = !pixTestHigh;
			wrap = true;
		}
		if (pixTestHigh)
		{
```

```
                if (wrap)
                        pixTestHigh = !pixTestHigh;
                if (testBaseCount >= freeCountPix && freeCountPix != -1)
                {
                        // allocation succesfull
                        return false;
                }
                else
                {
                        // not enough space in GPRs
                        return true;
                }
        }
        else
        {
                if (wrap)
                        pixTestHigh = !pixTestHigh;
                if (testBaseCount <= freeCountPix && freeCountPix != -1)
                {
                        // allocation succesfull
                        return false;
                }
                else
                {
                        return true;
                }
        }
}
// vertices
else
{
        testBaseCount = baseCountVert;
        base_addr= baseCountVert;

        // special case for MAX GPRs
        if (number_gpr == -(vertLimit-128))
        {
                if (freeCountVert==baseCountVert && vertTestHigh &&
                        freeCountVert != -1)
                {
                        return false;
                }
                else
                        return true;
        }

        if (testBaseCount - number_gpr >= vertLimit)
                testBaseCount = testBaseCount - number_gpr;
        else
        {
                testBaseCount = 128-(number_gpr-(testBaseCount-vertLimit));
                // we wrapped change the test type
                vertTestHigh = !vertTestHigh;
                wrap = true;
        }
        if (vertTestHigh)
        {
                if (wrap)
```

```
                        vertTestHigh = !vertTestHigh;
                if (testBaseCount <= freeCountVert && freeCountVert != -1)
                {
                        // allocation succesfull
                        return false;
                }
                else
                {
                        return true;
                }
        }
        else
        {
                if (wrap)
                        vertTestHigh = !vertTestHigh;
                if (testBaseCount >= freeCountVert && freeCountVert != -1)
                {
                        // allocation succesfull
                        return false;
                }
                else
                {
                        return true;
                }
        }
    }
}

void GPR_manager::allocate(int number_gpr,int &base_addr,
                                                        Shader_Type stype)
{
    if (stype == PIXEL)
    {
        base_addr = baseCountPix;

        // special case for MAX GPRs
        if (number_gpr == pixLimit)
        {
                freeCountPix = -1;
        }

        if (baseCountPix + number_gpr < pixLimit)
                baseCountPix = base_addr + number_gpr;
        else
        {
                baseCountPix = number_gpr-(pixLimit-base_addr);
                // we wrapped change the test type
                pixTestHigh = !pixTestHigh;
        }
    }
    // vertices
    else
    {
        base_addr = baseCountVert;

        // special case for MAX GPRs
        if (number_gpr == -(vertLimit-128))
        {
```

```
                freeCountVert = -1;
            }

            if (baseCountVert - number_gpr >= vertLimit)
                baseCountVert = base_addr - number_gpr;
            else
            {
                baseCountVert = 128-(number_gpr-(base_addr-vertLimit));
                // we wrapped change the test type
                vertTestHigh = !vertTestHigh;
            }
        }
}

void GPR_manager::deAllocate(int number_gpr,Shader_Type stype)
{
        switch (stype)
        {
        case PIXEL:
            // special case for MAX GPRs
            if (number_gpr == pixLimit)
            {
                baseCountPix = 0;
                freeCountPix = 0;
                pixTestHigh = true;
                break;
            }
            if (freeCountPix + number_gpr < pixLimit)
                freeCountPix += number_gpr;
            else
            {
                freeCountPix = number_gpr-(pixLimit-freeCountPix);
                // we wrapped change the test type
                pixTestHigh = !pixTestHigh;
            }
            break;
        case VERTEX:
            // special case for MAX GPRs
            if (number_gpr == -(vertLimit-128))
            {
                baseCountVert = 127;
                freeCountVert = 127;
                vertTestHigh = true;
                break;
            }
            if (freeCountVert - number_gpr > vertLimit)
                freeCountVert -= number_gpr;
            else
            {
                freeCountVert = 128-(number_gpr-(freeCountVert-vertLimit));
                // we wrapped change the test type
                vertTestHigh = !vertTestHigh;
            }
            break;
        };
}
```

```
Write data to the GPRs:
Sq_block_model.cpp
// write to the SP dummy interface
RegVect* values;

regFile[j]->GetValues(values,address);

interpData.Address[i]=i+base_ptr;
interpData.NumParams = interp_params;

for (int k=0;k<16;k++)
{

	interpData.InterpData[i][k][j].field[0]=values[k].field[0];

	interpData.InterpData[i][k][j].field[1]=values[k].field[1];

	interpData.InterpData[i][k][j].field[2]=values[k].field[2];

	interpData.InterpData[i][k][j].field[3]=values[k].field[3];
}
// increment the GPR address
if (address+1 < gpr_manager->pixLimit)
{
	address ++;
}
else
{
	address = 0;
}
```

sq_alu.cpp

```
	#include "user_block_model.h"
#include "sq_alu.h"
#include "sq_sp.h"
#include <iostream>
#include "Scalar_HW/mathen.h"

using namespace std;

//------------------------------------------------------------------------
SQ_ALU::SQ_ALU()
{
	CoissuedInstruction = true;
	mathScalar = new MathEn();

};

SQ_ALU::~SQ_ALU()
{
	delete mathScalar;
};

//----- This function represents the entry point to the ALU from the Sequencer----------
```

```
void SQ_ALU::Execute(RegFile* Reg, OutBuffer &ExportBuffer ,const CStore & Constants,uint32
SrcAAddr, uint32 SrcBAddr, uint32 SrcCAddr,uint32 DestAddr, uint32 ScalarDestAddr, AluInstruction
Instruction,
                                        unsigned int valids[], uint32 VectorIndex,SQ_SP* pSQ_SP,
                                        Shader_Type currentAluType,bool pred[],cUSER_BLOCK_SQ*
pSQ,int idAlu)
{
        int i;

        sq = pSQ;
        // fill the dummy interface
        SQ_SP_data SPData;
        static Constant constant[4];
        static int PMasks[4][4] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
        static int CMask[4] = {0,0,0,0};
        static int RAddr[4] = {0,0,0,0};
        static int WAddr[4] = {0,0,0,0};
        static bool REn[4] = {false,false,false,false};
        static bool WEn[4] = {false,false,false,false};

        SPData.Phase = VectorIndex;
        for (i=0;i<4;i++)
        {
                SPData.ConstantValue[i]=constant[VectorIndex].field[0];
                SPData.ExportValid[i]=valids[i];
                SPData.ExportWValid[i]=2;
                SPData.Valids[i] = PMasks[VectorIndex][i];
        }

        SPData.ExportCount=VectorIndex;
        SPData.ExportLast=0;
        SPData.CMask = CMask[VectorIndex];
        SPData.RdAddr = RAddr[VectorIndex];
        SPData.WrAddr = WAddr[VectorIndex];
        SPData.RdEnable = REn[VectorIndex];
        SPData.WrEnable = WEn[VectorIndex];
        SPData.IndexCnt = 0;
        SPData.SType = currentAluType;

        if (SPData.Phase == 0)
                SPData.InstStart = true;
        else
                SPData.InstStart = false;

        switch(VectorIndex)
        {
        case 0:
                SPData.Instruction = Instruction.SrcASel + (Instruction.SourceANegate << 2) +
                                                        (Instruction.SourceASwizzle << 4) +
                                                        ((Instruction.VectorResultPointer&0x3F)<<12);
                break;
        case 1:
                SPData.Instruction = Instruction.SrcBSel + (Instruction.SourceBNegate << 2) +
                                                        (Instruction.SourceBSwizzle << 4) +
                                                        ((Instruction.ScalarResultPointer&0x3F)<<12);
                break;
```

```
        case 2:
                SPData.Instruction = Instruction.SrcCSel + (Instruction.SourceCNegate << 2) +
                                                        (Instruction.SourceCSwizzle << 4);
                break;
        case 3:
                SPData.Instruction = Instruction.VectorOpcode + (Instruction.ScalarOpcode << 5)+
                                                        (Instruction.VectorClamp << 11) +
(Instruction.ScalarClamp << 12)+
                                                        (Instruction.VectorWriteMask << 13) +
(Instruction.ScalarWriteMask << 17);
                break;
        }

        // do all the static stuff for next turn
        if (Instruction.SrcASel)
                Constants.GetConstValue(constant[VectorIndex],SrcAAddr);
        else if (Instruction.SrcBSel)
                Constants.GetConstValue(constant[VectorIndex],SrcBAddr);
        else if (Instruction.SrcCSel)
                Constants.GetConstValue(constant[VectorIndex],SrcCAddr);

        for (i=0;i<4;i++)
                PMasks[VectorIndex][i] = valids[i];

        switch(VectorIndex)
        {
        case 0: // interpolator and SRC A
                CMask[VectorIndex] = 127-SrcAAddr;
                RAddr[VectorIndex] = SrcAAddr;
                WAddr[VectorIndex] = 126-SrcAAddr;
                REn[VectorIndex] = true;
                WEn[VectorIndex] = false;
                break;
        case 1: //TX and SRC B
      CMask[VectorIndex] = 125-SrcBAddr;
                RAddr[VectorIndex] = SrcBAddr;
                WAddr[VectorIndex] = 124-SrcBAddr;
                REn[VectorIndex] = true;
                WEn[VectorIndex] = false;
                break;
        case 2: // Vector and SRC C
                CMask[VectorIndex] = Instruction.VectorWriteMask;
                RAddr[VectorIndex] = SrcCAddr;
                REn[VectorIndex] = false; // no tree operands for now
                // if exporting
      if (((Instruction.VectorResultPointer & 0x80) != 0) && (Instruction.PredicateSelect < 2)) {
        WAddr[VectorIndex] = Instruction.VectorResultPointer & 0x3F;
        WEn[VectorIndex] = false;
      }
      else {
        WAddr[VectorIndex] = DestAddr;
        WEn[VectorIndex] = true;
      }
                break;
        case 3: // Scalar and TX
                CMask[VectorIndex] = Instruction.ScalarWriteMask;
```

```
                RAddr[VectorIndex] = 123-ScalarDestAddr;
                REn[VectorIndex] = false;
                // if exporting
      if (((Instruction.ScalarResultPointer & 0x80) != 0) && (Instruction.PredicateSelect < 2)) {
        WAddr[VectorIndex] = Instruction.ScalarResultPointer & 0x3F;
        WEn[VectorIndex] = false;
      }
                /*
      else {
        WAddr[VectorIndex] = ScalarDestAddr;
        WEn[VectorIndex] = true;
      }*/ // No scalar ops for now...
                break;
          }

          pSQ_SP->SetAll(&SPData);
          pSQ_SP->SetValid(true);

          // Real Emulator code
          CurrentRegFile = Reg;
          OutputBuffer = &ExportBuffer;
          CurrentAluInstruction = Instruction;
          AluPhase = VectorIndex;
          AluType = currentAluType;
          Predicates = &(pred[0]);
          validBits= &(valids[0]);
          AluId = idAlu;
          ExecuteAluInstruction(SrcAAddr,SrcBAddr,SrcCAddr,DestAddr,ScalarDestAddr,VectorIndex,Con
stants);

}
//---------------------------------------------------------------------------

void SQ_ALU::ExecuteAluInstruction(uint32 SrcAPtr, uint32 SrcBPtr, uint32 SrcCPtr, uint32 DstPtr,uint32
ScalarDestPtr,uint32 VectorIndex,const CStore & Constants)
{
          VectorData SrcA, SrcB, SrcC, VectorResult;
          mfloat<8,23,128> ScalarResult;
          VectorData TempSrc;

          bool error = false;

          const RegVect* InputVectorA;
          const RegVect* InputVectorB;
          const RegVect* InputVectorC;

          Constant ConstantA;
          Constant ConstantB;
          Constant ConstantC;


          RegisterFileRead(SrcAPtr,SrcBPtr,SrcCPtr,InputVectorA,InputVectorB,InputVectorC);

          //Going through all the 128bit vectors (16 of them)
          //They all have the same relative location inside their respective GPR files.
          for(uint8 vector_id = 0; vector_id <16 ; vector_id ++)
```

```
{

        SrcAReg.red     =InputVectorA[vector_id].field[0];
        SrcAReg.green =InputVectorA[vector_id].field[1];
        SrcAReg.blue    =InputVectorA[vector_id].field[2];
        SrcAReg.alpha  =InputVectorA[vector_id].field[3];

        SrcBReg.red     =InputVectorB[vector_id].field[0];
        SrcBReg.green =InputVectorB[vector_id].field[1];
        SrcBReg.blue    =InputVectorB[vector_id].field[2];
        SrcBReg.alpha  =InputVectorB[vector_id].field[3];

        SrcCReg.red     =InputVectorC[vector_id].field[0];
        SrcCReg.green =InputVectorC[vector_id].field[1];
        SrcCReg.blue    =InputVectorC[vector_id].field[2];
        SrcCReg.alpha  =InputVectorC[vector_id].field[3];

        // set the constants
        int cAddr =0;
        // relative addressing of the constant store via address register
        if (CurrentAluInstruction.SrcASel == 0 && CurrentAluInstruction.Constan0RelAbs == 1
&&
                CurrentAluInstruction.RelativeAddrRegSel == 1)
        {
                cAddr = SrcAPtr + ConstantOffsets[AluPhase*16+vector_id];
                if (AluType == VERTEX)
                {
                        if ((cAddr - sq->SQ_VS_CONST.getBASE())
                                > sq->SQ_VS_CONST.getSIZE())
                        {
                                cAddr = 0;
                                if (((validBits[vector_id/4])>>(vector_id%4))&0x01)
                                        error = true;
                        }
                }
                else
                {
                        if ((cAddr - sq->SQ_PS_CONST.getBASE())
                                > sq->SQ_PS_CONST.getSIZE())
                        {
                                cAddr = 0;
                                if (((validBits[vector_id/4])>>(vector_id%4))&0x01)
                                        error = true;
                        }
                }

                Constants.GetConstValue(ConstantA, cAddr);
        }
        else
                Constants.GetConstValue(ConstantA,SrcAPtr);

        // relative addressing of the constant store via address register
        if (((CurrentAluInstruction.SrcASel == 1 &&
                CurrentAluInstruction.SrcBSel == 0 && CurrentAluInstruction.Constan0RelAbs
== 1) ||
                (CurrentAluInstruction.SrcASel == 0 &&
```

```
                CurrentAluInstruction.SrcBSel == 0 && CurrentAluInstruction.Constan1RelAbs
== 1)) &&
                CurrentAluInstruction.RelativeAddrRegSel == 1)
        {
                cAddr = SrcBPtr + ConstantOffsets[AluPhase*16+vector_id];

                if (AluType == VERTEX)
                {
                        if ((cAddr - sq->SQ_VS_CONST.getBASE())
                                > sq->SQ_VS_CONST.getSIZE())
                        {
                                cAddr = 0;
                                if (((validBits[vector_id/4])>>(vector_id%4))&0x01)
                                        error = true;
                        }
                }
                else
                {
                        if ((cAddr - sq->SQ_PS_CONST.getBASE())
                                > sq->SQ_PS_CONST.getSIZE())
                        {
                                cAddr = 0;
                                if (((validBits[vector_id/4])>>(vector_id%4))&0x01)
                                        error = true;
                        }
                }

                Constants.GetConstValue(ConstantB, cAddr);
        }
        else
                Constants.GetConstValue(ConstantB, SrcBPtr);

        // relative addressing of the constant store via address register
        if (((CurrentAluInstruction.SrcASel == 1 &&
                CurrentAluInstruction.SrcBSel == 1 &&
                CurrentAluInstruction.SrcCSel == 0 && CurrentAluInstruction.Constan0RelAbs
== 1) ||
                ((CurrentAluInstruction.SrcASel == 0 ||
                CurrentAluInstruction.SrcBSel == 0) && CurrentAluInstruction.SrcCSel == 0
                && CurrentAluInstruction.Constan1RelAbs == 1)) &&
                CurrentAluInstruction.RelativeAddrRegSel == 1)
        {
                cAddr = SrcCPtr + ConstantOffsets[AluPhase*16+vector_id];

                if (AluType == VERTEX)
                {
                        if ((cAddr - sq->SQ_VS_CONST.getBASE())
                                > sq->SQ_VS_CONST.getSIZE())
                        {
                                cAddr = 0;
                                if (((validBits[vector_id/4])>>(vector_id%4))&0x01)
                                        error = true;
                        }
                }
                else
                {
```

```
                        if ((cAddr - sq->SQ_PS_CONST.getBASE())
                                > sq->SQ_PS_CONST.getSIZE())
                        {
                                cAddr = 0;
                                if (((validBits[vector_id/4])>>(vector_id%4))&0x01)
                                        error = true;
                        }
                }

                Constants.GetConstValue(ConstantC, cAddr);
        }
        else
                Constants.GetConstValue(ConstantC, SrcCPtr);

        // there was an addressing error
        if (error)
        {
                if (sq->SQ_DEBUG_MISC_0.getDB_PROB_ON())
                {
                        if (sq->SQ_DEBUG_MISC_0.getDB_PROB_COUNT() == 0)
                        {
                                sq->SQ_DEBUG_MISC_0.setDB_PROB_COUNT(1);
                                sq->SQ_DEBUG_MISC_0.setDB_PROB_ADDR(0);
                        }
                        else
                                sq->SQ_DEBUG_MISC_0.setDB_PROB_COUNT(sq-
>SQ_DEBUG_MISC_0.getDB_PROB_COUNT()+1);
                }
        }


        //muxing&swizzling&modification of input arguments
        //-----------------------------------------------
        uint32 SrcASel,SrcBSel,SrcCSel;
        SrcASel = CurrentAluInstruction.SrcASel;
        SrcBSel = CurrentAluInstruction.SrcBSel;
        SrcCSel = CurrentAluInstruction.SrcCSel;

        uint8 SrcASelRelAbs, SrcBSelRelAbs,SrcCSelRelAbs;
        SrcASelRelAbs = ((CurrentAluInstruction.SourceARegPointer)>>6) & 0x01;
        SrcBSelRelAbs = ((CurrentAluInstruction.SourceBRegPointer)>>6) & 0x01;
        SrcCSelRelAbs = ((CurrentAluInstruction.SourceCRegPointer)>>6) & 0x01;

        switch(SrcASel)
        {
        case NON_CONSTANT:
                switch(SrcASelRelAbs)
                {
                case ABSOLUTE_REG:
                case RELATIVE_REG:
                        SrcA.alpha = SrcAReg.alpha;
                        SrcA.red = SrcAReg.red;
                        SrcA.green = SrcAReg.green;
                        SrcA.blue = SrcAReg.blue;
                        break;
                default:
```

```
                break;
        }
        break;
case CONSTANT:
        SrcA.red = ConstantA.field[0];
        SrcA.green = ConstantA.field[1];
        SrcA.blue = ConstantA.field[2];
        SrcA.alpha = ConstantA.field[3];
        break;
}


switch(SrcBSel)
{
case NON_CONSTANT:
        switch(SrcBSelRelAbs)
        {
        case ABSOLUTE_REG:
        case RELATIVE_REG:
                SrcB.alpha = SrcBReg.alpha;
                SrcB.red = SrcBReg.red;
                SrcB.green = SrcBReg.green;
                SrcB.blue = SrcBReg.blue;
                break;
        default:
                break;
        }
        break;
case CONSTANT:
        SrcB.red = ConstantB.field[0];
        SrcB.green = ConstantB.field[1];
        SrcB.blue = ConstantB.field[2];
        SrcB.alpha = ConstantB.field[3];
        break;
}

switch(SrcCSel)
{
case NON_CONSTANT:
        switch(SrcCSelRelAbs)
        {
        case ABSOLUTE_REG:
        case RELATIVE_REG:
                SrcC.alpha = SrcCReg.alpha;
                SrcC.red = SrcCReg.red;
                SrcC.green = SrcCReg.green;
                SrcC.blue = SrcCReg.blue;
                break;
        default:
                break;
        }break;
case CONSTANT:
        SrcC.red = ConstantC.field[0];
        SrcC.green = ConstantC.field[1];
        SrcC.blue = ConstantC.field[2];
        SrcC.alpha = ConstantC.field[3];
```

```
        break;
}

//swizzling of arguments
uint8 SrcASwizzleAlpha = CurrentAluInstruction.SourceASwizzle >> 6;
uint8 SrcASwizzleBlue = (CurrentAluInstruction.SourceASwizzle >> 4)&0x3;
uint8 SrcASwizzleGreen = (CurrentAluInstruction.SourceASwizzle >>2)&0x3;
uint8 SrcASwizzleRed = (CurrentAluInstruction.SourceASwizzle)&0x3;

TempSrc.alpha = SrcA.alpha;
TempSrc.red = SrcA.red;
TempSrc.green =SrcA.green;
TempSrc.blue= SrcA.blue;

switch(SrcASwizzleAlpha)
{
case 0:break;
case 1:
        SrcA.alpha = TempSrc.red;
        break;
case 2:
        SrcA.alpha = TempSrc.green;
        break;
case 3:
        SrcA.alpha = TempSrc.blue;
        break;
}

switch(SrcASwizzleRed)
{
case 0:break;
case 1:
        SrcA.red = TempSrc.green;
        break;
case 2:
        SrcA.red = TempSrc.blue;
        break;
case 3:
        SrcA.red = TempSrc.alpha;
        break;
}

switch(SrcASwizzleGreen)
{
case 0:break;
case 1:
        SrcA.green = TempSrc.blue;
        break;
case 2:
        SrcA.green = TempSrc.alpha;
        break;
case 3:
        SrcA.green = TempSrc.red;
        break;
}
```

```
switch(SrcASwizzleBlue)
{
case 0:break;
case 1:
        SrcA.blue = TempSrc.alpha;
        break;
case 2:
        SrcA.blue = TempSrc.red;
        break;
case 3:
        SrcA.blue = TempSrc.green;
        break;
}
  //-----------------------------------------


TempSrc.alpha = SrcB.alpha;
TempSrc.red = SrcB.red;
TempSrc.green =SrcB.green;
TempSrc.blue= SrcB.blue;

uint8 SrcBSwizzleAlpha = (CurrentAluInstruction.SourceBSwizzle >> 6)&0x3;
uint8 SrcBSwizzleBlue = (CurrentAluInstruction.SourceBSwizzle >> 4)&0x3;
uint8 SrcBSwizzleGreen = (CurrentAluInstruction.SourceBSwizzle >>2)&0x3;
uint8 SrcBSwizzleRed = (CurrentAluInstruction.SourceBSwizzle)&0x3;

switch(SrcBSwizzleAlpha)
{
case 0:break;
case 1:
        SrcB.alpha = TempSrc.red;
        break;
case 2:
        SrcB.alpha = TempSrc.green;
        break;
case 3:
        SrcB.alpha = TempSrc.blue;
        break;
}

switch(SrcBSwizzleRed)
{
case 0:break;
case 1:
        SrcB.red = TempSrc.green;
        break;
case 2:
        SrcB.red = TempSrc.blue;
        break;
case 3:
        SrcB.red = TempSrc.alpha;
        break;
}

switch(SrcBSwizzleGreen)
```

```
{
case 0:break;
case 1:
        SrcB.green = TempSrc.blue;
        break;
case 2:
        SrcB.green = TempSrc.alpha;
        break;
case 3:
        SrcB.green = TempSrc.red;
        break;
}


switch(SrcBSwizzleBlue)
{
case 0:break;
case 1:
        SrcB.blue = TempSrc.alpha;
        break;
case 2:
        SrcB.blue = TempSrc.red;
        break;
case 3:
        SrcB.blue = TempSrc.green;
        break;
}

//-------------------------------------------------------

TempSrc.alpha = SrcC.alpha;
TempSrc.red = SrcC.red;
TempSrc.green =SrcC.green;
TempSrc.blue= SrcC.blue;

uint8 SrcCSwizzleAlpha = CurrentAluInstruction.SourceCSwizzle >> 6;
uint8 SrcCSwizzleBlue = (CurrentAluInstruction.SourceCSwizzle >> 4)&0x3;
uint8 SrcCSwizzleGreen = (CurrentAluInstruction.SourceCSwizzle >>2 )&0x3;
uint8 SrcCSwizzleRed = (CurrentAluInstruction.SourceCSwizzle)&0x3;

switch(SrcCSwizzleAlpha)
{
case 0:break;
case 1:
        SrcC.alpha = TempSrc.red;
        break;
case 2:
        SrcC.alpha = TempSrc.green;
        break;
case 3:
        SrcC.alpha = TempSrc.blue;
        break;
}


switch(SrcCSwizzleRed)
```

```
{
case 0:break;
case 1:
        SrcC.red = TempSrc.green;
        break;
case 2:
        SrcC.red = TempSrc.blue;
        break;
case 3:
        SrcC.red = TempSrc.alpha;
        break;
}

switch(SrcCSwizzleGreen)
{
case 0:break;
case 1:
        SrcC.green = TempSrc.blue;
        break;
case 2:
        SrcC.green = TempSrc.alpha;
        break;
case 3:
        SrcC.green = TempSrc.red;
        break;
}


switch(SrcCSwizzleBlue)
{
case 0:break;
case 1:
        SrcC.blue = TempSrc.alpha;
        break;
case 2:
        SrcC.blue = TempSrc.red;
        break;
case 3:
        SrcC.blue = TempSrc.green;
        break;
}

// ABS MODIFIER
uint8 SrcAAbs = (CurrentAluInstruction.SourceARegPointer>>7)&0x01;
uint8 SrcBAbs = (CurrentAluInstruction.SourceBRegPointer>>7)&0x01;
uint8 SrcCAbs = (CurrentAluInstruction.SourceCRegPointer>>7)&0x01;
uint8 Cst0Abs = (CurrentAluInstruction.VectorResultPointer>>7)&0x01;

if (SrcASel == NON_CONSTANT)
{
        switch (SrcAAbs){
        case NO_ABS_MOD:
                break;
        case ABS_MOD:
                SrcA.red.abs();
                SrcA.green.abs();
```

```
                SrcA.blue.abs();
                SrcA.alpha.abs();
                break;
        default:
                break;
        };
}
else
{
        switch (Cst0Abs){
        case NO_ABS_MOD:
                break;
        case ABS_MOD:
                SrcA.red.abs();
                SrcA.green.abs();
                SrcA.blue.abs();
                SrcA.alpha.abs();
                break;
        default:
                break;
        };
}

if (SrcBSel == NON_CONSTANT)
{
        switch (SrcBAbs){
        case NO_ABS_MOD:
                break;
        case ABS_MOD:
                SrcB.red.abs();
                SrcB.green.abs();
                SrcB.blue.abs();
                SrcB.alpha.abs();
                break;
        default:
                break;
        };
}
else if (SrcBSel == CONSTANT)
{
        switch (Cst0Abs){
        case NO_ABS_MOD:
                break;
        case ABS_MOD:
                SrcB.red.abs();
                SrcB.green.abs();
                SrcB.blue.abs();
                SrcB.alpha.abs();
                break;
        default:
                break;
        };
}

if (SrcCSel == NON_CONSTANT)
{
```

```
        switch (SrcCAbs){
        case NO_ABS_MOD:
                break;
        case ABS_MOD:
                SrcC.red.abs();
                SrcC.green.abs();
                SrcC.blue.abs();
                SrcC.alpha.abs();

                PreviousScalar[AluId][VectorIndex][vector_id].alpha.abs();
                PreviousScalar[AluId][VectorIndex][vector_id].red.abs();
                PreviousScalar[AluId][VectorIndex][vector_id].green.abs();
                PreviousScalar[AluId][VectorIndex][vector_id].blue.abs();
                break;
        default:
                break;
        };
}
else if (SrcCSel == CONSTANT)
{
        switch (Cst0Abs){
        case NO_ABS_MOD:
                break;
        case ABS_MOD:
                SrcC.red.abs();
                SrcC.green.abs();
                SrcC.blue.abs();
                SrcC.alpha.abs();

                PreviousScalar[AluId][VectorIndex][vector_id].alpha.abs();
                PreviousScalar[AluId][VectorIndex][vector_id].red.abs();
                PreviousScalar[AluId][VectorIndex][vector_id].green.abs();
                PreviousScalar[AluId][VectorIndex][vector_id].blue.abs();
                break;
        default:
                break;
        };
}

//---------------------------------------------
//negate input modifiers
uint8 SrcANegate= CurrentAluInstruction.SourceANegate;
uint8 SrcBNegate= CurrentAluInstruction.SourceBNegate;
uint8 SrcCNegate= CurrentAluInstruction.SourceCNegate;

switch(SrcANegate){
case NIL:break;
case NEGATE:
        SrcA.alpha.neg();
        SrcA.red.neg();
        SrcA.green.neg();
        SrcA.blue.neg();
        break;
default:
        break;
}
```

```
switch(SrcBNegate){
case NIL:break;
case NEGATE:
        SrcB.alpha.neg();
        SrcB.red.neg();
        SrcB.green.neg();
        SrcB.blue.neg();
        break;
default:
        break;
}

switch(SrcCNegate){
case NIL:break;
case NEGATE:
        SrcC.alpha.neg();
        SrcC.red.neg();
        SrcC.green.neg();
        SrcC.blue.neg();

        PreviousScalar[AluId][VectorIndex][vector_id].alpha.neg();
        PreviousScalar[AluId][VectorIndex][vector_id].red.neg();
        PreviousScalar[AluId][VectorIndex][vector_id].green.neg();
        PreviousScalar[AluId][VectorIndex][vector_id].blue.neg();
        break;
default:
        break;
}

//-------------------------------------------------------
//Execute ALU opcode
ExecuteAluOpcode(SrcA,SrcB,SrcC,VectorResult,ScalarResult,vector_id);

// Clamp results if told to
VectorResult.red = Clamp(VectorResult.red,true);
VectorResult.green = Clamp(VectorResult.green,true);
VectorResult.blue = Clamp(VectorResult.blue,true);
VectorResult.alpha = Clamp(VectorResult.alpha,true);

ScalarResult = Clamp(ScalarResult,false);

//Save Previous Vector and Scalar
PreviousVector[AluId][VectorIndex][vector_id].alpha = VectorResult.alpha;
PreviousVector[AluId][VectorIndex][vector_id].red = VectorResult.red;
PreviousVector[AluId][VectorIndex][vector_id].green = VectorResult.green;
PreviousVector[AluId][VectorIndex][vector_id].blue = VectorResult.blue;

PreviousScalar[AluId][VectorIndex][vector_id].alpha = ScalarResult;
PreviousScalar[AluId][VectorIndex][vector_id].red = ScalarResult;
PreviousScalar[AluId][VectorIndex][vector_id].green = ScalarResult;
PreviousScalar[AluId][VectorIndex][vector_id].blue = ScalarResult;


//-------------------------------------------------------
```

```
		//Accumulate the result into an array of 16x128
		VectorVector.Val[vector_id].field[0] =VectorResult.red;
		VectorVector.Val[vector_id].field[1] =VectorResult.green;
		VectorVector.Val[vector_id].field[2] =VectorResult.blue;
		VectorVector.Val[vector_id].field[3] =VectorResult.alpha ;


		ScalarVector.Val[vector_id].field[0] =ScalarResult;
		ScalarVector.Val[vector_id].field[1] =ScalarResult;
		ScalarVector.Val[vector_id].field[2] =ScalarResult;
		ScalarVector.Val[vector_id].field[3] =ScalarResult;

		//-----------------------------------------------------
		//Exporting the results
		bool Export = (CurrentAluInstruction.ScalarResultPointer>>7)&0x1;

		if(Export)
		{
			// fog exports
			if (((CurrentAluInstruction.VectorResultPointer&0x3F) >= 16) &&
				((CurrentAluInstruction.VectorResultPointer&0x3F) < 20) &&
				(CurrentAluInstruction.VectorWriteMask&0x01) &&
(CurrentAluInstruction.ScalarWriteMask&0x01))
			{
				unsigned int inVect;
				unsigned int inFog;
				unsigned int blended;

				// RED
				float value = VectorResult.red.getReal();
				inVect = *(reinterpret_cast<unsigned int*>(&value));
				value = ScalarResult.getReal();
				inFog = *(reinterpret_cast<unsigned int*>(&value));
				inFog = inFog >> 8;

				blended = (inVect) | (inFog&0x3F);
				value = *(reinterpret_cast<float*>(&blended));

				// export blended red
				OutputBuffer->values[vector_id].field[0] = value;

				// GREEN
				value = VectorResult.green.getReal();
				inVect = *(reinterpret_cast<unsigned int*>(&value));
				blended = (inVect) | ((inFog>>6)&0x3F);
				value = *(reinterpret_cast<float*>(&blended));

				// export blended green
				OutputBuffer->values[vector_id].field[1] = value;


				// BLUE
				value = VectorResult.blue.getReal();
				inVect = *(reinterpret_cast<unsigned int*>(&value));
				blended = (inVect) | ((inFog>>12)&0x3F);
				value = *(reinterpret_cast<float*>(&blended));
```

```
                        // export blended blue
                        OutputBuffer->values[vector_id].field[2] = value;


                        // ALPHA
                        value = VectorResult.alpha.getReal();
                        inVect = *(reinterpret_cast<unsigned int*>(&value));
                        blended = (inVect) | ((inFog>>18)&0x3F);
                        value = *(reinterpret_cast<float*>(&blended));

                        // export blended alpha
                        OutputBuffer->values[vector_id].field[3] = value;
                }
                else
                {
                        // RED COMPONENT
                        if (CurrentAluInstruction.VectorWriteMask&0x01 &&
CurrentAluInstruction.ScalarWriteMask&0x01)
                                OutputBuffer->values[vector_id].field[0] = 1.0;
                        else if (CurrentAluInstruction.VectorWriteMask&0x01)
                                OutputBuffer->values[vector_id].field[0] = VectorResult.red;
                        else if (CurrentAluInstruction.ScalarWriteMask&0x01)
                                OutputBuffer->values[vector_id].field[0] = ScalarResult;
                        // GREEN COMPONENT
                        if ((CurrentAluInstruction.VectorWriteMask>>1)&0x01 &&
(CurrentAluInstruction.ScalarWriteMask>>1)&0x01)
                                OutputBuffer->values[vector_id].field[1] = 1.0;
                        else if ((CurrentAluInstruction.VectorWriteMask>>1)&0x01)
                                OutputBuffer->values[vector_id].field[1] = VectorResult.green;
                        else if ((CurrentAluInstruction.ScalarWriteMask>>1)&0x01)
                                OutputBuffer->values[vector_id].field[1] = ScalarResult;
                        // BLUE COMPONENT
                        if ((CurrentAluInstruction.VectorWriteMask>>2)&0x01 &&
(CurrentAluInstruction.ScalarWriteMask>>2)&0x01)
                                OutputBuffer->values[vector_id].field[2] = 1.0;
                        else if ((CurrentAluInstruction.VectorWriteMask>>2)&0x01)
                                OutputBuffer->values[vector_id].field[2] = VectorResult.blue;
                        else if ((CurrentAluInstruction.ScalarWriteMask>>2)&0x01)
                                OutputBuffer->values[vector_id].field[2] = ScalarResult;
                        // ALPHA COMPONENT
                        if ((CurrentAluInstruction.VectorWriteMask>>3)&0x01 &&
(CurrentAluInstruction.ScalarWriteMask>>3)&0x01)
                                OutputBuffer->values[vector_id].field[3] = 1.0;
                        else if ((CurrentAluInstruction.VectorWriteMask>>3)&0x01)
                                OutputBuffer->values[vector_id].field[3] = VectorResult.alpha;
                        else if ((CurrentAluInstruction.ScalarWriteMask>>3)&0x01)
                                OutputBuffer->values[vector_id].field[3] = ScalarResult;
                }

                // predicate the exports here
                int predValid;
                int predicat;
                int j;
                for (int i=0;i<4;i++)
                {
```

```
                                predValid = validBits[i];
                                predicat = 0;

                                if (CurrentAluInstruction.PredicateSelect == 2)
                                {
                                        for (j=0;j<4;j++)
                                                predicat += (!(Predicates[i*4+j]))<<j;
                                        predValid &= predicat;
                                }
                                else if (CurrentAluInstruction.PredicateSelect == 3)
                                {
                                        for (j=0;j<4;j++)
                                                predicat += Predicates[i*4+j]<<j;
                                        predValid &= predicat;
                                }

                                OutputBuffer->valids[i]=predValid;
                        }
                        OutputBuffer->valid = true;
                }
        }
                //write the result into register files
        RegisterFileWrite(CurrentAluInstruction.VectorWriteMask,CurrentAluInstruction.ScalarWriteMask,
                ScalarDestPtr,DstPtr);
}

//-----------------------------------------------------------------------------------------------

void SQ_ALU::ExecuteAluOpcode(VectorData SrcA, VectorData SrcB, VectorData SrcC, VectorData &
VectorResult,mfloat<8,23,128> & ScalarResult, int i)
{
        mfloat<8,23,128> red;
        mfloat<8,23,128> green;
        mfloat<8,23,128> blue;
        mfloat<8,23,128> alpha;

        mfloat<8,23,128> one;
        one.putReal((float)1.0);
        mfloat<8,23,128> zero;
        zero.putReal((float)0.0);
        mfloat<8,23,128> two;
        two.putReal((float)2.0);

        CoissuedInstruction = true;

        //Executing Vector Opcode
        switch(CurrentAluInstruction.VectorOpcode)
        {
        case ADDv:
          {
            if(sq->isHardwareAccurate())
            {
                                VectorResult.alpha = multiply_add(SrcA.alpha,one,SrcB.alpha);
                                VectorResult.red = multiply_add(SrcA.red,one,SrcB.red);
                                VectorResult.green = multiply_add(SrcA.green,one,SrcB.green);
                                VectorResult.blue = multiply_add(SrcA.blue,one,SrcB.blue);
```

```
    }
    else
    {
                        VectorResult.alpha.add(SrcA.alpha,SrcB.alpha);
                        VectorResult.red.add(SrcA.red,SrcB.red);
                        VectorResult.green.add(SrcA.green,SrcB.green);
                        VectorResult.blue.add(SrcA.blue,SrcB.blue);

    }
    break;
  }
case MAXv:
        VectorResult.alpha.max(SrcA.alpha,SrcB.alpha);
        VectorResult.red.max(SrcA.red,SrcB.red);
        VectorResult.green.max(SrcA.green,SrcB.green);
        VectorResult.blue.max(SrcA.blue,SrcB.blue);
        break;
case MINv:
        VectorResult.alpha.min(SrcA.alpha,SrcB.alpha);
        VectorResult.red.min(SrcA.red,SrcB.red);
        VectorResult.green.min(SrcA.green,SrcB.green);
        VectorResult.blue.min(SrcA.blue,SrcB.blue);
        break;
case MULv:
  if(sq->isHardwareAccurate())
  {
     VectorResult.alpha = multiply_add(SrcA.alpha, SrcB.alpha,zero);
     VectorResult.red = multiply_add(SrcA.red, SrcB.red,zero);
     VectorResult.green = multiply_add(SrcA.green, SrcB.green,zero);
     VectorResult.blue = multiply_add(SrcA.blue, SrcB.blue,zero);
  }
  else
  {
     VectorResult.alpha.mul(SrcA.alpha,SrcB.alpha);
     VectorResult.red.mul(SrcA.red,SrcB.red);
     VectorResult.green.mul(SrcA.green,SrcB.green);
     VectorResult.blue.mul(SrcA.blue,SrcB.blue);
  }
  break;
case SETEv:
        VectorResult.alpha = (SrcA.alpha == SrcB.alpha) ? 1.0:0.0;
        VectorResult.red = (SrcA.red == SrcB.red) ? 1.0:0.0;
        VectorResult.green = (SrcA.green == SrcB.green) ? 1.0:0.0;
        VectorResult.blue = (SrcA.blue == SrcB.blue) ? 1.0:0.0;
        break;
case SETGTv:
        VectorResult.alpha = (SrcA.alpha > SrcB.alpha) ? 1.0:0.0;
        VectorResult.red = (SrcA.red > SrcB.red) ? 1.0:0.0;
        VectorResult.green = (SrcA.green > SrcB.green) ? 1.0:0.0;
        VectorResult.blue = (SrcA.blue > SrcB.blue) ? 1.0:0.0;
        break;
case SETGTEv:
        VectorResult.alpha = (SrcA.alpha >= SrcB.alpha) ? 1.0:0.0;
        VectorResult.red = (SrcA.red >= SrcB.red) ? 1.0:0.0;
        VectorResult.green = (SrcA.green >= SrcB.green) ? 1.0:0.0;
        VectorResult.blue = (SrcA.blue >= SrcB.blue) ? 1.0:0.0;
```

```
        break;
case SETNEv:
        VectorResult.alpha = (SrcA.alpha != SrcB.alpha) ? 1.0:0.0;
        VectorResult.red = (SrcA.red != SrcB.red) ? 1.0:0.0;
        VectorResult.green = (SrcA.green != SrcB.green) ? 1.0:0.0;
        VectorResult.blue = (SrcA.blue != SrcB.blue) ? 1.0:0.0;
        break;
case FRACv:
        VectorResult.alpha.sub(SrcA.alpha,(float)((int)SrcA.alpha.getReal()));
        VectorResult.red.sub(SrcA.red,(float)((int)SrcA.red.getReal()));
        VectorResult.green.sub(SrcA.green,(float)((int)SrcA.green.getReal()));
        VectorResult.blue.sub(SrcA.blue,(float)((int)SrcA.blue.getReal()));
        break;
case TRUNCv:
        VectorResult.alpha = (float)((int)SrcA.alpha.getReal());
        VectorResult.red = (float)((int)SrcA.red.getReal());
        VectorResult.green = (float)((int)SrcA.green.getReal());
        VectorResult.blue = (float)((int)SrcA.blue.getReal());
        break;
case FLOORv:
        if (SrcA.alpha.getReal() >= 0)
                VectorResult.alpha = (float)((int)SrcA.alpha.getReal());
        else
                VectorResult.alpha = (float)((int)SrcA.alpha.getReal())-1.0f;
        if (SrcA.red.getReal() >= 0)
                VectorResult.red = (float)((int)SrcA.red.getReal());
        else
                VectorResult.red = (float)((int)SrcA.red.getReal())-1.0f;
        if (SrcA.green.getReal() >= 0)
                VectorResult.green = (float)((int)SrcA.green.getReal());
        else
                VectorResult.green = (float)((int)SrcA.green.getReal())-1.0f;
        if (SrcA.blue.getReal() >= 0)
                VectorResult.blue = (float)((int)SrcA.blue.getReal());
        else
                VectorResult.blue = (float)((int)SrcA.blue.getReal())-1.0f;
        break;
case MULADDv:
   if(sq->isHardwareAccurate())
   {
                        VectorResult.alpha = multiply_add(SrcA.alpha, SrcB.alpha,SrcC.alpha);
                        VectorResult.red = multiply_add(SrcA.red, SrcB.red,SrcC.red);
                        VectorResult.green = multiply_add(SrcA.green, SrcB.green,SrcC.green);
                        VectorResult.blue = multiply_add(SrcA.blue, SrcB.blue,SrcC.blue);
   }
   else
   {
                        VectorResult.alpha.mad(SrcA.alpha,SrcB.alpha,SrcC.alpha);
                        VectorResult.red.mad(SrcA.red,SrcB.red,SrcC.red);
                        VectorResult.green.mad(SrcA.green,SrcB.green,SrcC.green);
                        VectorResult.blue.mad(SrcA.blue,SrcB.blue,SrcC.blue);
   }
   CoissuedInstruction = false;
   break;
case DOT4v:
        if(sq->isHardwareAccurate())
```

```
    {
                VectorResult.alpha = multiply_add(SrcA.alpha, SrcB.alpha,zero);
                VectorResult.red = multiply_add(SrcA.red, SrcB.red,zero);
                VectorResult.green = multiply_add(SrcA.green, SrcB.green,zero);
                VectorResult.blue = multiply_add(SrcA.blue, SrcB.blue,zero);

                VectorResult.alpha = multiply_add(one,VectorResult.alpha,VectorResult.red);
                VectorResult.alpha = multiply_add(one,VectorResult.alpha,VectorResult.green);
                VectorResult.alpha = multiply_add(one,VectorResult.alpha,VectorResult.blue);
                VectorResult.red = VectorResult.alpha;
                VectorResult.green = VectorResult.alpha;
                VectorResult.blue = VectorResult.alpha;
        }
        else
        {
                alpha.mul(SrcA.alpha, SrcB.alpha);
                red.mul(SrcA.red, SrcB.red);
                green.mul(SrcA.green, SrcB.green);
                blue.mul(SrcA.blue, SrcB.blue);

                VectorResult.alpha.add(alpha,red);
                VectorResult.alpha +=green;
                VectorResult.alpha +=blue;
                VectorResult.red = VectorResult.alpha;
                VectorResult.green = VectorResult.alpha;
                VectorResult.blue = VectorResult.alpha;
        }
        break;
case DOT3v:
        if(sq->isHardwareAccurate())
    {
                VectorResult.red = multiply_add(SrcA.red, SrcB.red,zero);
                VectorResult.green = multiply_add(SrcA.green, SrcB.green,zero);
                VectorResult.blue = multiply_add(SrcA.blue, SrcB.blue,zero);

                VectorResult.red = multiply_add(one,VectorResult.red,VectorResult.green);
                VectorResult.red = multiply_add(one,VectorResult.red,VectorResult.blue);
                VectorResult.green = VectorResult.red;
                VectorResult.blue = VectorResult.red;
                VectorResult.alpha = VectorResult.red;
        }
        else
        {
                red.mul(SrcA.red,SrcB.red);
                green.mul(SrcA.green, SrcB.green);
                blue.mul(SrcA.blue,SrcB.blue);
                VectorResult.red.add(red,green);
                VectorResult.red += blue;
                VectorResult.alpha = VectorResult.red;
                VectorResult.green = VectorResult.red;
                VectorResult.blue = VectorResult.red;
        }
        break;
case CNDEv:
        VectorResult.alpha = (SrcA.alpha == 0.0) ? SrcB.alpha:SrcC.alpha;
        VectorResult.red = (SrcA.red == 0.0) ? SrcB.red:SrcC.red;
```

```
        VectorResult.green = (SrcA.green == 0.0) ? SrcB.green:SrcC.green;
        VectorResult.blue = (SrcA.blue == 0.0) ? SrcB.blue:SrcC.blue;
        break;
case CNDGTv:
        VectorResult.alpha = (SrcA.alpha > 0.0) ? SrcB.alpha:SrcC.alpha;
        VectorResult.red = (SrcA.red > 0.0) ? SrcB.red:SrcC.red;
        VectorResult.green = (SrcA.green > 0.0) ? SrcB.green:SrcC.green;
        VectorResult.blue = (SrcA.blue > 0.0) ? SrcB.blue:SrcC.blue;
        break;
case CNDGTEv:
        VectorResult.alpha = (SrcA.alpha >= 0.0) ? SrcB.alpha:SrcC.alpha;
        VectorResult.red = (SrcA.red >= 0.0) ? SrcB.red:SrcC.red;
        VectorResult.green = (SrcA.green >= 0.0) ? SrcB.green:SrcC.green;
        VectorResult.blue = (SrcA.blue >= 0.0) ? SrcB.blue:SrcC.blue;
        break;
case CUBEv:
        if (SrcA.red > SrcA.green && SrcA.red > SrcA.blue)
        {
                VectorResult.red = SrcA.red;
                if (SrcA.red >= 0)
                {
                        VectorResult.green =0;
                        VectorResult.alpha = -SrcA.blue;
                        VectorResult.blue = -SrcA.green;
                }
                else
                {
                        VectorResult.green =1;
                        VectorResult.alpha = SrcA.blue;
                        VectorResult.blue = -SrcA.green;
                }
        }
        else if (SrcA.green > SrcA.blue)
        {
                VectorResult.red = SrcA.green;
                if (SrcA.green >= 0)
                {
                        VectorResult.green =2;
                        VectorResult.alpha = SrcA.red;
                        VectorResult.blue = SrcA.blue;
                }
                else
                {
                        VectorResult.green =3;
                        VectorResult.alpha = SrcA.red;
                        VectorResult.blue = -SrcA.blue;
                }
        }
        else
        {
                VectorResult.red = SrcA.blue;
                if (SrcA.blue >= 0)
                {
                        VectorResult.green =4;
                        VectorResult.alpha = SrcA.red;
                        VectorResult.blue = -SrcA.green;
```

```
                }
                else
                {
                        VectorResult.green =5;
                        VectorResult.alpha = -SrcA.red;
                        VectorResult.blue = -SrcA.green;
                }
        }
        if(sq->isHardwareAccurate())
  {
                VectorResult.red = multiply_add(VectorResult.red,two,zero);
        }
        else
        {
                VectorResult.red.mul(2,VectorResult.red);
        }
        break;
case MAX4v:
        if (SrcA.red > SrcA.green && SrcA.red > SrcA.blue && SrcA.red > SrcA.alpha)
                VectorResult.alpha = SrcA.red;
        else if (SrcA.green > SrcA.blue && SrcA.green > SrcA.alpha)
                VectorResult.alpha = SrcA.green;
        else if (SrcA.blue > SrcA.alpha)
                VectorResult.alpha = SrcA.blue;
        else
                VectorResult.alpha = SrcA.alpha;

        VectorResult.red = VectorResult.alpha;
        VectorResult.green = VectorResult.alpha;
        VectorResult.blue = VectorResult.alpha;
        break;
case DOT2ADDv:
        if(sq->isHardwareAccurate())
  {
                VectorResult.red = multiply_add(SrcA.red, SrcB.red,zero);
                VectorResult.green = multiply_add(SrcA.green, SrcB.green,zero);

                VectorResult.red = multiply_add(one,VectorResult.red,VectorResult.green);
                VectorResult.red = multiply_add(one,VectorResult.red,SrcC.red);
                VectorResult.alpha = VectorResult.red;
                VectorResult.green = VectorResult.red;
                VectorResult.blue = VectorResult.red;
        }
        else
        {
                VectorResult.red.mul(SrcA.red,SrcB.red);
                VectorResult.green.mul(SrcA.green,SrcB.green);
                VectorResult.red.add(VectorResult.red,VectorResult.green);
                VectorResult.red.add(VectorResult.red,SrcC.red);

                VectorResult.alpha = VectorResult.red;
                VectorResult.green = VectorResult.red;
                VectorResult.blue = VectorResult.red;
        }
        break;
case PRED_SETE_PUSHv:
```

```
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcB.alpha.getReal() == 0.0f && SrcA.red.getReal() == 0.0f)
                {
                        Predicates[i] = true;
                        VectorResult.red = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        VectorResult.red = SrcA.red.getReal()+1.0f;
                }
        }
        break;
case PRED_SETGT_PUSHv:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcB.alpha.getReal() > 0.0f && SrcA.red.getReal() == 0.0f)
                {
                        Predicates[i] = true;
                        VectorResult.red = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        VectorResult.red = SrcA.red.getReal()+1.0f;
                }
        }
        break;
case PRED_SETGTE_PUSHv:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcB.alpha.getReal() >= 0.0f && SrcA.red.getReal() == 0.0f)
                {
                        Predicates[i] = true;
                        VectorResult.red = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        VectorResult.red = SrcA.red.getReal()+1.0f;
                }
        }
        break;
case PRED_SETNE_PUSHv:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcB.alpha.getReal() != 0.0f && SrcA.red.getReal() == 0.0f)
```

```
                        {
                                Predicates[i] = true;
                                VectorResult.red = 0.0f;
                        }
                        else
                        {
                                Predicates[i] = false;
                                VectorResult.red = SrcA.red.getReal()+1.0f;
                        }
                }
                break;
        case KILLEv:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                {
                        if (SrcA.alpha.getReal() == SrcB.alpha.getReal() && SrcA.red.getReal() ==
SrcB.red.getReal() &&
                                SrcA.green.getReal() == SrcB.green.getReal() && SrcA.blue.getReal()
== SrcB.blue.getReal())
                        {
                                validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                        }
                }
                break;
        case KILLGTv:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                {
                        if (SrcA.alpha.getReal() > SrcB.alpha.getReal() && SrcA.red.getReal() >
SrcB.red.getReal() &&
                                SrcA.green.getReal() > SrcB.green.getReal() && SrcA.blue.getReal() >
SrcB.blue.getReal())
                        {
                                validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                        }
                }
                break;
        case KILLGTEv:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                {
                        if (SrcA.alpha.getReal() >= SrcB.alpha.getReal() && SrcA.red.getReal() >=
SrcB.red.getReal() &&
                                SrcA.green.getReal() >= SrcB.green.getReal() && SrcA.blue.getReal()
>= SrcB.blue.getReal())
                        {
                                validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                        }
                }
                break;
        case KILLNEv:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
```

```
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                {
                        if (SrcA.alpha.getReal() != SrcB.alpha.getReal() && SrcA.red.getReal() !=
SrcB.red.getReal() &&
                                SrcA.green.getReal() != SrcB.green.getReal() && SrcA.blue.getReal() !=
SrcB.blue.getReal())
                        {
                                validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                        }
                }
                break;
        case MOVAv:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                {
                        ConstantOffsets[i+AluPhase*16] = floor(SrcA.alpha.getReal()+0.5);
                }
                VectorResult.red = SrcA.red;
                VectorResult.green = SrcA.green;
                VectorResult.blue = SrcA.blue;
                VectorResult.alpha = SrcA.alpha;
                break;
        case DSTv:
                VectorResult.red = 1.0f;
                if(sq->isHardwareAccurate())
                        VectorResult.green = multiply_add(SrcA.green, SrcB.green,zero);
                else
                        VectorResult.green.mul(SrcA.green,SrcB.green);
                VectorResult.blue = SrcA.blue;
                VectorResult.alpha = SrcB.alpha;
                break;
        default:
                std::cerr << "Unsuported Vector Opcode in SP: " << CurrentAluInstruction.VectorOpcode
<< std::endl;
        }

        //Executing Scalar Opcode
        //Note: There is a coissue only when vector opcode uses two sources or less

        nanCheck nanValue;
        Vector4 result,in;

        if(CoissuedInstruction)
        {
                switch(CurrentAluInstruction.ScalarOpcode)
                {
                case ADDs:
                        if(sq->isHardwareAccurate())
                                ScalarResult = multiply_add(SrcC.alpha,one,SrcC.red);
                        else
                                ScalarResult.add(SrcC.alpha,SrcC.red);
                        break;
                case ADD_PREVs:
                        if(sq->isHardwareAccurate())
```

```
                                ScalarResult =
multiply_add(SrcC.alpha,one,PreviousScalar[AluId][AluPhase][i].red);
                        else
                                ScalarResult.add(SrcC.alpha,PreviousScalar[AluId][AluPhase][i].red);
                        break;
                case MULs:
                        if(sq->isHardwareAccurate())
                                ScalarResult = multiply_add(SrcC.alpha,SrcC.red,zero);
                        else
                                ScalarResult.mul(SrcC.alpha,SrcC.red);
                        break;
                case MUL_PREVs:
                        if(sq->isHardwareAccurate())
                                ScalarResult =
multiply_add(SrcC.alpha,PreviousScalar[AluId][AluPhase][i].red,zero);
                        else
                                ScalarResult.mul(SrcC.alpha,PreviousScalar[AluId][AluPhase][i].red);
                        break;
                case MUL_PREV2s:
                        nanValue.f = PreviousScalar[AluId][AluPhase][i].red.getReal();
                        if (nanValue.u == 0xFF7FFFFF || nanValue.u == 0xFF800000 ||
                                SrcC.red.getReal() <= 0)
                        {
                                nanValue.u = 0xFF7FFFFF;
                                ScalarResult = nanValue.f;
                        }
                        else
                        {
                                if(sq->isHardwareAccurate())
                                        ScalarResult =
multiply_add(SrcC.alpha,PreviousScalar[AluId][AluPhase][i].red,zero);
                                else

        ScalarResult.mul(SrcC.alpha,PreviousScalar[AluId][AluPhase][i].red);
                        }
                        break;
                case MINs:
                        ScalarResult.min(SrcC.alpha,SrcC.red);
                        break;
                case MAXs:
                        ScalarResult.max(SrcC.alpha,SrcC.red);
                        break;
                case SETEs:
                        ScalarResult = (SrcC.alpha == 0.0) ? 1.0:0.0;
                        break;
                case SETNEs:
                        ScalarResult = (SrcC.alpha != 0.0) ? 1.0:0.0;
                        break;
                case SETGTs:
                        ScalarResult = (SrcC.alpha > 0.0) ? 1.0:0.0;
                        break;
                case SETGTEs:
                        ScalarResult= (SrcC.alpha >= 0.0) ? 1.0:0.0;
                        break;
                case FRACs:
                        ScalarResult.sub(SrcC.alpha,(float)((int)SrcC.alpha.getReal()));
```

```
        break;
case TRUNCs:
        ScalarResult= (float)((int)SrcC.alpha.getReal());
        break;
case FLOORs:
        if (SrcC.alpha.getReal() > 0)
                ScalarResult = (float)((int)SrcC.alpha.getReal());
        else
                ScalarResult = (float)((int)SrcC.alpha.getReal())-1.0f;
        break;
case EXP_IEEE:
        nanValue.f = SrcC.alpha.getReal();
        // 0
        if (SrcC.alpha.getReal() == 0.0f)
        {
                ScalarResult = 1.0f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
                ScalarResult = nanValue.f;
        }
        // + INF
        else if (nanValue.u == 0x7F800000)
        {
                ScalarResult = nanValue.f;
        }
        // - INF
        else if (nanValue.u == 0xFF800000)
        {
                ScalarResult = 0.0f;
        }
        // + MAX_FLT
        else if (nanValue.u == 0x7F7FFFFF)
        {
                nanValue.u = 0x7F800000;
                ScalarResult = nanValue.f;
        }
        // - MAX_FLT
        else if (nanValue.u == 0xFF7FFFFF)
        {
                ScalarResult = 0.0f;
        }
        else
        {
                if(sq->isHardwareAccurate())
                {
                        in.x = SrcC.alpha.getReal();
                        mathScalar->ExpBase2FullDX4(&result.x,&in.x);
                        ScalarResult = result.x;
                }
                else
                {
                        ScalarResult = pow(2,SrcC.alpha.getReal());
                }
        }
```

```
        break;
case LOG_CLAMP:
        nanValue.f = SrcC.alpha.getReal();
        // 0
        if (SrcC.alpha.getReal() == 0.0f)
        {
                nanValue.u = 0xFF7FFFFF;
                ScalarResult = nanValue.f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
                ScalarResult = nanValue.f;
        }
        // + INF
        else if (nanValue.u == 0x7F800000)
        {
                ScalarResult = nanValue.f;
        }
        // - INF
        else if (nanValue.u == 0xFF800000)
        {
                nanValue.u = R400_NAN;
                ScalarResult = nanValue.f;
        }
        // neg
        else if (nanValue.f < 0)
        {
                nanValue.u = R400_NAN;
                ScalarResult = nanValue.f;
        }
        else
        {
                if(sq->isHardwareAccurate())
                {
                        in.x = SrcC.alpha.getReal();
                        mathScalar->LogBase2FullDX4(&result.x,&in.x);
                        ScalarResult = result.x;
                }
                else
                {
                        ScalarResult = log(SrcC.alpha.getReal())/log(2);
                }
        }
        break;
case LOG_IEEE:
        nanValue.f = SrcC.alpha.getReal();
        // 0
        if (SrcC.alpha.getReal() == 0.0f)
        {
                nanValue.u = 0xFF800000;
                ScalarResult = nanValue.f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
```

```
                ScalarResult = nanValue.f;
            }
            // + INF
            else if (nanValue.u == 0x7F800000)
            {
                ScalarResult = nanValue.f;
            }
            // - INF
            else if (nanValue.u == 0xFF800000)
            {
                nanValue.u = R400_NAN;
                ScalarResult = nanValue.f;
            }
            // neg
            else if (nanValue.f < 0)
            {
                nanValue.u = R400_NAN;
                ScalarResult = nanValue.f;
            }
            else
            {
                if(sq->isHardwareAccurate())
                {
                    in.x = SrcC.alpha.getReal();
                    mathScalar->LogBase2FullDX4(&result.x,&in.x);
                    ScalarResult = result.x;
                }
                else
                {
                    ScalarResult = log(SrcC.alpha.getReal())/log(2);
                }
            }
            break;
        case RECIP_CLAMP:
            nanValue.f = SrcC.alpha.getReal();
            // + 0
            if (nanValue.u == 0x00000000)
            {
                nanValue.u = 0x7F7FFFFF;
                ScalarResult = nanValue.f;
            }
            // - 0
            else if (nanValue.u == 0x80000000)
            {
                nanValue.u = 0xFF7FFFFF;
                ScalarResult = nanValue.f;
            }
            // NAN
            else if (nanValue.f != nanValue.f)
            {
                ScalarResult = nanValue.f;
            }
            // + INF
            else if (nanValue.u == 0x7F800000)
            {
                nanValue.u = 0x80000000;
```

```
                ScalarResult = nanValue.f;
        }
        // - INF
        else if (nanValue.u == 0xFF800000)
        {
                nanValue.u = 0x00000000;
                ScalarResult = nanValue.f;
        }
        if(sq->isHardwareAccurate())
        {
                in.x = SrcC.alpha.getReal();
                mathScalar->RecipFF(&result.x,&in.x);
                ScalarResult = result.x;
        }
        else
                ScalarResult.div(1.0,SrcC.alpha);
        break;
case RECIP_FF:
        nanValue.f = SrcC.alpha.getReal();
        // + 0
        if (nanValue.u == 0x00000000)
        {
                nanValue.u = 0x00000000;
                ScalarResult = nanValue.f;
        }
        // - 0
        else if (nanValue.u == 0x80000000)
        {
                nanValue.u = 0x80000000;
                ScalarResult = nanValue.f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
                ScalarResult = nanValue.f;
        }
        // + INF
        else if (nanValue.u == 0x7F800000)
        {
                nanValue.u = 0x80000000;
                ScalarResult = nanValue.f;
        }
        // - INF
        else if (nanValue.u == 0xFF800000)
        {
                nanValue.u = 0x00000000;
                ScalarResult = nanValue.f;
        }
        else
        {
                if(sq->isHardwareAccurate())
                {
                        in.x = SrcC.alpha.getReal();
                        mathScalar->RecipFF(&result.x,&in.x);
                        ScalarResult = result.x;
                }
```

```
                else
                        ScalarResult.div(1.0,SrcC.alpha);
        }
        break;
case RECIP_IEEE:
        nanValue.f = SrcC.alpha.getReal();
        // + 0
        if (nanValue.u == 0x00000000)
        {
                nanValue.u = 0x7F800000;
                ScalarResult = nanValue.f;
        }
        // - 0
        else if (nanValue.u == 0x80000000)
        {
                nanValue.u = 0xFF800000;
                ScalarResult = nanValue.f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
                ScalarResult = nanValue.f;
        }
        // + INF
        else if (nanValue.u == 0x7F800000)
        {
                nanValue.u = 0x80000000;
                ScalarResult = nanValue.f;
        }
        // - INF
        else if (nanValue.u == 0xFF800000)
        {
                nanValue.u = 0x00000000;
                ScalarResult = nanValue.f;
        }
        else
        {
                if(sq->isHardwareAccurate())
                {
                        in.x = SrcC.alpha.getReal();
                        mathScalar->RecipFF(&result.x,&in.x);
                        ScalarResult = result.x;
                }
                else
                        ScalarResult.div(1.0,SrcC.alpha);
        }
        break;
case RECIPSQ_CLAMP:
        nanValue.f = SrcC.alpha.getReal();
        // + 0
        if (nanValue.u == 0x00000000)
        {
                nanValue.u = 0x7F7FFFFF;
                ScalarResult = nanValue.f;
        }
        // - 0
```

```
        else if (nanValue.u == 0x80000000)
        {
                nanValue.u = 0xFF7FFFFF;
                ScalarResult = nanValue.f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
                ScalarResult = nanValue.f;
        }
        // + INF
        else if (nanValue.u == 0x7F800000)
        {
                nanValue.u = 0x00000000;
                ScalarResult = nanValue.f;
        }
        // - INF
        else if (nanValue.u == 0xFF800000)
        {
                nanValue.u = R400_NAN;
                ScalarResult = nanValue.f;
        }
        // -
        else if (nanValue.f < 0.0f)
        {
                nanValue.u = R400_NAN;
                ScalarResult = nanValue.f;
        }
        if(sq->isHardwareAccurate())
        {
                in.x = SrcC.alpha.getReal();
                mathScalar->RecipSqrtFF(&result.x,&in.x);
                ScalarResult = result.x;
        }
        else
                ScalarResult = sqrt(ScalarResult.div(1.0,SrcC.alpha).getReal());
        break;
case RECIPSQ_FF:
        nanValue.f = SrcC.alpha.getReal();
        // + 0
        if (nanValue.u == 0x00000000)
        {
                nanValue.u = 0x00000000;
                ScalarResult = nanValue.f;
        }
        // - 0
        else if (nanValue.u == 0x80000000)
        {
                nanValue.u = 0x80000000;
                ScalarResult = nanValue.f;
        }
        // NAN
        else if (nanValue.f != nanValue.f)
        {
                ScalarResult = nanValue.f;
        }
```

```
		// + INF
		else if (nanValue.u == 0x7F800000)
		{
			nanValue.u = 0x00000000;
			ScalarResult = nanValue.f;
		}
		// - INF
		else if (nanValue.u == 0xFF800000)
		{
			nanValue.u = R400_NAN;
			ScalarResult = nanValue.f;
		}
		// -
		else if (nanValue.f < 0.0f)
		{
			nanValue.u = R400_NAN;
			ScalarResult = nanValue.f;
		}
		else
		{
			if(sq->isHardwareAccurate())
			{
				in.x = SrcC.alpha.getReal();
				mathScalar->RecipSqrtFF(&result.x,&in.x);
				ScalarResult = result.x;
			}
			else
				ScalarResult = sqrt(ScalarResult.div(1.0,SrcC.alpha).getReal());
		}
		break;
	case RECIPSQ_IEEE:
		nanValue.f = SrcC.alpha.getReal();
		// + 0
		if (nanValue.u == 0x00000000)
		{
			nanValue.u = 0x7F800000;
			ScalarResult = nanValue.f;
		}
		// - 0
		else if (nanValue.u == 0x80000000)
		{
			nanValue.u = 0xFF800000;
			ScalarResult = nanValue.f;
		}
		// NAN
		else if (nanValue.f != nanValue.f)
		{
			ScalarResult = nanValue.f;
		}
		// + INF
		else if (nanValue.u == 0x7F800000)
		{
			nanValue.u = 0x00000000;
			ScalarResult = nanValue.f;
		}
		// - INF
```

```
                else if (nanValue.u == 0xFF800000)
                {
                        nanValue.u = R400_NAN;
                        ScalarResult = nanValue.f;
                }
                // -
                else if (nanValue.f < 0.0f)
                {
                        nanValue.u = R400_NAN;
                        ScalarResult = nanValue.f;
                }
                else
                {
                        if(sq->isHardwareAccurate())
                        {
                                in.x = SrcC.alpha.getReal();
                                mathScalar->RecipSqrtFF(&result.x,&in.x);
                                ScalarResult = result.x;
                        }
                        else
                                ScalarResult = sqrt(ScalarResult.div(1.0,SrcC.alpha).getReal());
                }
                break;
        case MOVAs:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                        ConstantOffsets[i+AluPhase*16] = floor(SrcC.alpha.getReal()+0.5);
                ScalarResult = SrcC.alpha;
                break;
        case MOVA_FLOORs:
                // check for predication
                if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                        (CurrentAluInstruction.PredicateSelect>>1) == 0)
                        ConstantOffsets[i+AluPhase*16] = floor(SrcC.alpha.getReal());
                ScalarResult = SrcC.alpha;
                break;
        case SUBs:
                if(sq->isHardwareAccurate())
                {
                        green = -1.0f;
                        ScalarResult = multiply_add(SrcC.red,green,SrcC.alpha);
                }
                else
                        ScalarResult.sub(SrcC.alpha,SrcC.red);
                break;
        case SUB_PREVs:
                if(sq->isHardwareAccurate())
                {
                        green = -1.0f;
                        ScalarResult =
multiply_add(PreviousScalar[AluId][AluPhase][i].red,green,SrcC.alpha);
                }
                else
                        ScalarResult.sub(SrcC.alpha,PreviousScalar[AluId][AluPhase][i].red);
                break;
```

```
case PRED_SETEs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.alpha.getReal() == 0.0f)
                {
                        Predicates[i]= true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        ScalarResult = 1.0f;
                }
        }
        break;
case PRED_SETGTs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.alpha.getReal() > 0.0f)
                {
                        Predicates[i]= true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        ScalarResult = 1.0f;
                }
        }
        break;
case PRED_SETGTEs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.alpha.getReal() >= 0.0f)
                {
                        Predicates[i]= true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        ScalarResult = 1.0f;
                }
        }
        break;
case PRED_SETNEs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
```

```
                if (SrcC.alpha.getReal() != 0.0f)
                {
                        Predicates[i]= true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        ScalarResult = 1.0f;
                }
        }
        break;
case PRED_SET_INVs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.red.getReal() == 1.0f)
                {
                        Predicates[i]= true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        if (SrcC.red.getReal() == 0.0f)
                                ScalarResult = 1.0f;
                        else
                                ScalarResult = SrcC.red.getReal();
                }
        }
        break;
case PRED_SET_POPs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.red.getReal()-1.0f <= 0.0f)
                {
                        Predicates[i]= true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        ScalarResult = SrcC.red.getReal()-1.0f;
                }
        }
        break;
case PRED_SET_CLRs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                Predicates[i] = false;
                // set to max float
```

```
                nanValue.u = 0x7F7FFFFF;
                ScalarResult = nanValue.f;
        }
        break;
case PRED_SET_RESTOREs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.red.getReal() == 0.0f)
                {
                        Predicates[i] = true;
                        ScalarResult = 0.0f;
                }
                else
                {
                        Predicates[i] = false;
                        ScalarResult = SrcC.red.getReal();
                }
        }
        break;
case KILLEs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.alpha.getReal() == 0.0f)
                {
                        validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                }
        }
        break;
case KILLGTs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.alpha.getReal() > 0.0f)
                {
                        validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                }
        }
        break;
case KILLGTEs:
        // check for predication
        if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
                (CurrentAluInstruction.PredicateSelect>>1) == 0)
        {
                if (SrcC.alpha.getReal() >= 0.0f)
                {
                        validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
                }
        }
        break;
case KILLNEs:
        // check for predication
```

```
		if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
			(CurrentAluInstruction.PredicateSelect>>1) == 0)
		{
			if (SrcC.alpha.getReal() != 0.0f)
			{
				validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
			}
		}
		break;
	case KILLONEs:
		// check for predication
		if ((CurrentAluInstruction.PredicateSelect&0x01) == Predicates[i] ||
			(CurrentAluInstruction.PredicateSelect>>1) == 0)
		{
			if (SrcC.alpha.getReal() == 1.0f)
			{
				validBits[i/4] = validBits[i/4]&(0xEF>>(4-(i%4)));
			}
		}
		break;
	case SQRT_IEEE:
		nanValue.f = SrcC.alpha.getReal();
		// + 0
		if (nanValue.u == 0x00000000)
		{
			nanValue.u = 0x00000000;
			ScalarResult = nanValue.f;
		}
		// - 0
		else if (nanValue.u == 0x80000000)
		{
			nanValue.u = 0x80000000;
			ScalarResult = nanValue.f;
		}
		// NAN
		else if (nanValue.f != nanValue.f)
		{
			ScalarResult = nanValue.f;
		}
		// + INF
		else if (nanValue.u == 0x7F800000)
		{
			nanValue.u = 0x7F800000;
			ScalarResult = nanValue.f;
		}
		// - INF
		else if (nanValue.u == 0xFF800000)
		{
			nanValue.u = R400_NAN;
			ScalarResult = nanValue.f;
		}
		// -
		else if (nanValue.f < 0.0f)
		{
			nanValue.u = R400_NAN;
			ScalarResult = nanValue.f;
```

```
                        }
                        else
                                ScalarResult = pow(2,0.5*log(SrcC.alpha.getReal())/log(2));
                        break;
                default:
                        std::cerr << "Scalar Opcode Not supported: " <<
((int)CurrentAluInstruction.ScalarOpcode) << std::endl;
                        break;
                }
        }
}
//---------------------------------------------------------------------------
void SQ_ALU::RegisterFileRead(uint32 SrcAPtr,uint32 SrcBPtr,uint32 SrcCPtr,const RegVect*
&InputVectorA,
                                                        const RegVect* &InputVectorB,const RegVect*
&InputVectorC)
{
        CurrentRegFile->GetConstValues(InputVectorA,SrcAPtr);
        CurrentRegFile->GetConstValues(InputVectorB,SrcBPtr);
        CurrentRegFile->GetConstValues(InputVectorC,SrcCPtr);
}
//---------------------------------------------------------------------------
void SQ_ALU:: RegisterFileWrite(uint8 VectorWriteMask, uint8 ScalarWriteMask,uint32 ScalarAddr,
                                                                uint32 VectorAddr)
{

        //grabing a pointer to the GPR entry in location VectorAddr
        RegVect* CurrentRegEntry;
        CurrentRegFile->GetValues(CurrentRegEntry, VectorAddr);

        // if not exporting
        if (!((CurrentAluInstruction.ScalarResultPointer>>7)&0x1))
        {
                if (VectorWriteMask != 0x0)
                {
                        for (int vector_id = 0; vector_id < 16 ; vector_id ++){
                                for( int channel = 0; channel < 4 ; channel ++){
                                        if (VectorWriteMask&(1<<channel))
                                                if ((CurrentAluInstruction.PredicateSelect&0x01) ==
Predicates[vector_id] ||
                                                        (CurrentAluInstruction.PredicateSelect>>1) ==
0)
                                                        CurrentRegEntry[vector_id].field[channel] =
VectorVector.Val[vector_id].field[channel];
                                }
                        }
                }
        }

        //grabing a pointer to the GPR entry in location ScalarAddr
        CurrentRegFile->GetValues(CurrentRegEntry, ScalarAddr);
        // if not exporting
        if (!((CurrentAluInstruction.ScalarResultPointer>>7)&0x1))
        {
                if (ScalarWriteMask != 0x0)
                {
```

```
                        for (int vector_id = 0; vector_id < 16 ; vector_id ++){
                                for( int channel = 0; channel < 4 ; channel ++){
                                if (ScalarWriteMask&(1<<channel))
                                        if ((CurrentAluInstruction.PredicateSelect&0x01) ==
Predicates[vector_id] ||
                                                                (CurrentAluInstruction.PredicateSelect>>1) ==
0)
                                                                CurrentRegEntry[vector_id].field[channel] =
ScalarVector.Val[vector_id].field[channel];
                                        }
                                }
                        }
                }
}

//-------------------------------------------------------------------------------------------------

mfloat<8,23,128> SQ_ALU::Clamp(mfloat<8,23,128> result, bool vector)
{
        mfloat<8,23,128> one;
        one.putReal((float)1.0);
        mfloat<8,23,128> zero;
   zero.putReal((float)0.0);
        mfloat<8,23,128> result_clamped ;

        if(vector){
                result_clamped = ((result > one) & CurrentAluInstruction.VectorClamp) ? one: result;
                result_clamped = ((result < zero) & CurrentAluInstruction.VectorClamp) ? zero :
result_clamped;
        }
        else{
                result_clamped = ((result > one) & CurrentAluInstruction.ScalarClamp) ? one : result;
                result_clamped = ((result < zero) & CurrentAluInstruction.ScalarClamp) ? zero :
result_clamped;
        }
        return (result_clamped);

}
```

Sq_block_model.cpp

```
#include <chip/ar_code/ar_chip_interface.h>
#include <gfx/sq/user_block_model.h>
#include <gfx/sx/user_block_model.h>
#include <gfx/sq/sq_dumps.h>
#include <sys/rom/user_block_model.h>
#include <reg/crayola_offset.h>
#include <numbers/numbers.h>
#include <tex/tconst.h>
#include <core/registry.h>
```

```
#include <iostream>
#include <queue>

#include "reg_file.h"
#include "sq_alu.h"
#include "constant_store.h"
#include "interpolators.h"
#include "instruction_store.h"
#include "arbiter.h"
#include "alu_types.h"

//#define DEBUG_SEQ

using namespace std;

//--------------------------------------------------------------------------
cUSER_BLOCK_SQ::cUSER_BLOCK_SQ (cAR_CHIP_INTERFACE*
                                                    pchip,      uint32      block_id,
cMODEL_BLOCK_PARAMETERS& blockParameters) :
cAR_BLOCK_SQ (pchip, block_id, blockParameters), interpolators(parameters)
{
        int i,j;
        std::cout << "block SQ constructor" << std::endl;

        #ifndef MSDOS
    m_dumpSQ = Core::Registry::read("HKEY_LOCAL_MACHINE\\SOFTWARE\\ATI
Technologies\\Debug\\SqDump", 0);
        #else
    m_dumpSQ = (uint32)(getenv("SqDump"));
        #endif // End MSDOS

        pSC_SQ=NULL;
        pSC_SP=NULL;
        pSQ_SC=NULL;
        pVGT_SQ_Verts=NULL;
        pVGT_SQ_verts_ready=NULL;
        pSQ_SP_Interp=NULL;
        pSQ_SX=NULL;
        pSP_SX=NULL;
        pSQ_TP=NULL;
        pSX_SQ=NULL;
        pSQ_SP=NULL;
        pTP_SQ=NULL;
        pSQ_CP_PIX =NULL;
        pSQ_CP_VTX = NULL;
        pSQ_RB = NULL;
```

```
        regFile[0]=NULL;
        regFile[1]=NULL;
        regFile[2]=NULL;
        regFile[3]=NULL;
        arbiter=NULL;
        gpr_manager=NULL;
        m_sqTpDump   = NULL;
        m_spSxDump   = NULL;
        m_sqSxDump  = NULL;

        idle0 = idle1_7 = 0;

        if(m_dumpSQ>0) {
    m_sqTpDump   = new cSqTp_Dump("sq_tp.dmp");
                m_spSxDump   = new cSpSx_Dump("sp_sx.dmp");
                m_sqSxDump   = new cSqSx_Dump("sq_sx.dmp");
                m_sqScDump   = new cSqSc_Dump("sq_sc.dmp");
                m_sqSpInterpDump   = new cSqSpInterp_Dump("sq_sp_interp.dmp");
                pcFile = fopen("sq_sx_pc.dmp","wb");
}

        auto_count_pix = 0;
        auto_count_vtx = 0;

        // set up the register files
        for (i=0;i<4;i++)
                regFile[i]= new RegFile();

        // clean the output buffer
        outBuffer.valid = false;
        for (i=0;i<16;i++)
        {
                outBuffer.values[i].field[0]=0.0;
                outBuffer.values[i].field[1]=0.0;
                outBuffer.values[i].field[2]=0.0;
                outBuffer.values[i].field[3]=0.0;
        }

        // ORDER IS RGBA A in [3] B in [2] G in [1] and R in [0] OR
// ORDER IS XYZW W in [3] Z in [2] Y in [1] and X in [0]
        // init the parameter store to all 0s
        for (j=0;j<16;j++)
        {
                for (i=0;i<128;i++)
                {
                        parameters[i].Val[j].field[0] = 0.0;
```

```
                parameters[i].Val[j].field[1] = 0.0;
                parameters[i].Val[j].field[2] = 0.0;
                parameters[i].Val[j].field[3] = 0.0;
        }
}

// clean the pixel input buffer
for (j=0;j<4;j++)
{
        interp[j].new_vector = false;
        interp[j].pc_dealloc = 0;
        interp[j].state_id = 0;
}

// clear the vertex shader ready counts
for (i=0;i<8;i++)
{
        vertexReady[i]=0;
}

for (i=0;i<64;i++)
        for (j=0;j<2;j++)
        {
                stagingRegisters[i][j].field[0] = 0.0f;
                stagingRegisters[i][j].field[1] = 0.0f;
                stagingRegisters[i][j].field[2] = 0.0f;
                stagingRegisters[i][j].field[3] = 0.0f;
        }

for (i=0;i<3;i++)
        for (j=0;j<16;j++)
        {
                RTParameters[i][j].field[0] = 0.0f;
                RTParameters[i][j].field[1] = 0.0f;
                RTParameters[i][j].field[2] = 0.0f;
                RTParameters[i][j].field[3] = 0.0f;
        }

// set the parameter cache head to 0
pcHead = 0;
// set the parameter cache head to 127
pcFree = 127;
// set the test type
checkHigh = true;

// create the ALU arbiter
```

```
        arbiter = new Arbiter(this,m_dumpSQ);

        // create the GPR manager
        gpr_manager = new GPR_manager(this);

        current_write_state = 0;
}

void cUSER_BLOCK_SQ::Reset()
{
        int i,j;
        for (i=0;i<4;i++)
                delete regFile[i];

        delete arbiter;
        delete gpr_manager;

        regFile[0]=NULL;
        regFile[1]=NULL;
        regFile[2]=NULL;
        regFile[3]=NULL;
        arbiter=NULL;
        gpr_manager=NULL;

        idle0 = idle1_7 = 0;

        auto_count_pix = 0;
        auto_count_vtx = 0;

        // set up the register files
        for (i=0;i<4;i++)
                regFile[i]= new RegFile();

        // clean the output buffer
        outBuffer.valid = false;
        for (i=0;i<16;i++)
        {
                outBuffer.values[i].field[0]=0.0;
                outBuffer.values[i].field[1]=0.0;
                outBuffer.values[i].field[2]=0.0;
                outBuffer.values[i].field[3]=0.0;
        }

        // ORDER IS RGBA A in [3] B in [2] G in [1] and R in [0] OR
    // ORDER IS XYZW W in [3] Z in [2] Y in [1] and X in [0]
        // init the parameter store to all 0s
```

```
for (j=0;j<16;j++)
{
        for (i=0;i<128;i++)
        {
                parameters[i].Val[j].field[0] = 0.0;
                parameters[i].Val[j].field[1] = 0.0;
                parameters[i].Val[j].field[2] = 0.0;
                parameters[i].Val[j].field[3] = 0.0;
        }
}

// clean the pixel input buffer
for (j=0;j<4;j++)
{
        interp[j].new_vector = false;
        interp[j].pc_dealloc = 0;
        interp[j].state_id = 0;
}

// clear the vertex shader ready counts
for (i=0;i<8;i++)
{
        vertexReady[i]=0;
}

for (i=0;i<64;i++)
        for (j=0;j<2;j++)
        {
                stagingRegisters[i][j].field[0] = 0.0f;
                stagingRegisters[i][j].field[1] = 0.0f;
                stagingRegisters[i][j].field[2] = 0.0f;
                stagingRegisters[i][j].field[3] = 0.0f;
        }

for (i=0;i<3;i++)
        for (j=0;j<16;j++)
        {
                RTParameters[i][j].field[0] = 0.0f;
                RTParameters[i][j].field[1] = 0.0f;
                RTParameters[i][j].field[2] = 0.0f;
                RTParameters[i][j].field[3] = 0.0f;
        }

// set the parameter cache head to 0
pcHead = 0;
// set the parameter cache head to 127
```

```
            pcFree = 127;
            // set the test type
            checkHigh = true;

            // create the ALU arbiter
            arbiter = new Arbiter(this,m_dumpSQ);

            // create the GPR manager
            gpr_manager = new GPR_manager(this);

            current_write_state = 0;
        }

        cUSER_BLOCK_SQ::~cUSER_BLOCK_SQ(void)
        {
             int i;
             for (i=0;i<4;i++)
                  delete regFile[i];

             if(m_dumpSQ>0) {
           delete(m_sqTpDump);
           delete(m_spSxDump);
           delete(m_sqSxDump);
                     delete(m_sqScDump);
                     delete(m_sqSpInterpDump);
                     fprintf(pcFile,"END\n");
                     fclose(pcFile);
          }
             delete arbiter;
             delete gpr_manager;
        }

        //*********************************************************************
*******
        // Main function for block
        //*********************************************************************
*******
        void cUSER_BLOCK_SQ::Main()
        {
           Fetch();
             Process();
             Output();
        }
        //*********************************************************************
*******
        // Fetch function for block
```

```
//*****************************************************************************
*******
void cUSER_BLOCK_SQ::Fetch(void)
{
   static sq_indx_count = 0;

      // grab the output of the PA and copy it localy
      pSC_SQ->GetAll(&sc_sq_data);
      pSC_SP->GetAll(&sc_sp_data);

      // grab the output of the VGT and copy it localy
      pVGT_SQ_Verts->GetAll(&vgt_sq_verts_data);

      if(!pVGT_SQ_verts_ready->GetReady())
            vgt_sq_verts_data.VGT_SQ_send = false;
#if 0
   if ( vgt_sq_verts_data.VGT_SQ_send && vgt_sq_verts_data.VGT_SQ_indx_valid ) {
      sq_indx_count++;
   }
   if ( vgt_sq_verts_data.VGT_SQ_send &&
vgt_sq_verts_data.VGT_SQ_end_of_vtx_vect ) {
      printf("sq_block_model: eov -- received %d real indices from
VGT\n",sq_indx_count);
      fflush(stdout);
      sq_indx_count = 0;
   }
#endif


      // ok for more new stuff
   pVGT_SQ_verts_ready->SetReady(true);

   // invalidate the TP interface
   pSQ_TP->SetValid(false);

   // invalidate SX interfaces
   pSQ_SX->SetValid(false);
   pSQ_SX->SetSQ_SX_exp_valid(false);
   pSQ_SX->SetSQ_SX_free_done(false);
   pSP_SX->SetValid(false);

   // invalidate SP interface
   pSQ_SP->SetValid(false);

   // invalidate CP interfaces
   pSQ_CP_VTX->SetValid(false);
```

```
pSQ_CP_PIX->SetValid(false);

// invalidate SP interface
pSQ_SP_Interp->SetValid(false);

// invalidate SC interface
pSQ_SC->SetSQ_SC_dec_cntr_cnt(false);
pSQ_SC->SetSQ_SC_free_buf(false);

// TEXTURE PIPE INTERFACE READ
static int phase = 0;
// grab the return from the texture pipe if valid
if (pTP_SQ->GetValid())
{
        TXColor returnColor;
        uinteger<7> registerAddress;
        RegVect* txAddr;
        int valid;

        registerAddress = pTP_SQ->GetTP_SP_gpr_dst();
        regFile[phase]->GetValues(txAddr,registerAddress);

        // Here we write the data to the GPRs. We only write data that has a
        // write mask activated
        for (int i=0;i<16;i++)
        {
                returnColor = pTP_SQ->GetTP_SP_data(i);
                valid = pTP_SQ->GetTP_SP_pix_mask(i/4).getValue();
                if ((valid>>(i%4))&0x01)
                {
                        if (pTP_SQ->GetTP_SP_cmask(0))
                                txAddr[i].field[0]=returnColor.x;
                        if (pTP_SQ->GetTP_SP_cmask(1))
                                txAddr[i].field[1]=returnColor.y;
                        if (pTP_SQ->GetTP_SP_cmask(2))
                                txAddr[i].field[2]=returnColor.z;
                        if (pTP_SQ->GetTP_SP_cmask(3))
                                txAddr[i].field[3]=returnColor.w;
                }
        }

        // increment the phase
        phase ++;

        if (phase == 4)
        {
```

```
                                phase =0;
                                // all texture instrucions of the clause have returned we can place
                                // the vector back in the next RS
                                if (pTP_SQ->GetTP_SQ_data_rdy())
                                {
                                        // set the ready flag in the RS
                                        if (pTP_SQ->GetTP_SQ_type() == VERTEX)
                                        {
                                                arbiter->vertexStation[pTP_SQ-
>GetTP_SQ_thread_id()].status.texReadsOutstanding = false;
                                        }
                                        else
                                        {
                                                arbiter->pixelStation[pTP_SQ-
>GetTP_SQ_thread_id()].status.texReadsOutstanding = false;
                                        }
                                }
                        }

                }
        }


        //*************************************************************************
*******
        // Process pixels function for block
        //*************************************************************************
*******
        void cUSER_BLOCK_SQ::ProcessPixels(void)
        {
                int i,j;
                int deallocating = 0;

                int ready = 0;

                static bool first_transfert = true;
                static int buf_read = 0;
                static int lineSQ[4] = {0,0,0,0};
                static int lineSP[4] = {0,0,0,0};
                static int SQ_buf_id = 0;

                static int QWrote = 0;
                bool pulsed = false;
                PixInputs pix;

                // first deal with these one clock transfers
```

```
		if (sc_sq_data.SC_SQ_event && sc_sq_data.SC_SQ_valid)
		{
			// filter out all events but fot the PS_DEALLOC and PS_TS_DEALLOC
			if (sc_sq_data.SC_SQ_event_id == PS_DEALLOC ||
sc_sq_data.SC_SQ_event_id == PS_DONE_TS
				|| sc_sq_data.SC_SQ_event_id == RST_PIX_CNT)
			{
				pix.event = sc_sq_data.SC_SQ_event_id;
				pix.state = sc_sq_data.SC_SQ_state_id;
				eventFIFO.push(pix);
				if (pix.state == 0)
					idle0 ++;
				else
					idle1_7 ++;
			}
			pSQ_SC->SetSQ_SC_dec_cntr_cnt(true);
			pulsed = true;
		}
		// new vector and dealloc tokens (without any other data)
		else if (first_transfert && sc_sq_data.SC_SQ_quad_mask[0] == 0
			&& sc_sq_data.SC_SQ_quad_mask[1] == 0 &&
sc_sq_data.SC_SQ_quad_mask[2] == 0 &&
			sc_sq_data.SC_SQ_quad_mask[3] == 0 && sc_sq_data.SC_SQ_valid)
		{
			if (sc_sq_data.SC_SQ_pc_dealloc > 0)
			{
				pix.event = 200+sc_sq_data.SC_SQ_pc_dealloc;
				pix.state = sc_sq_data.SC_SQ_state_id;
				eventFIFO.push(pix);
				pSQ_SC->SetSQ_SC_dec_cntr_cnt(true);
				pulsed = true;
				if (pix.state == 0)
					idle0 ++;
				else
					idle1_7 ++;
			}
			if (sc_sq_data.SC_SQ_new_vector)
			{
				pSQ_SC->SetSQ_SC_dec_cntr_cnt(true);
				pix.event = 300;
				pix.state = sc_sq_data.SC_SQ_state_id;
				eventFIFO.push(pix);
				pulsed = true;
				if (pix.state == 0)
					idle0 ++;
				else
```

```
                        idle1_7 ++;
                }
        }
        // accumulate the control data if something sent by the SC
        else if (sc_sq_data.SC_SQ_valid)
        {
                if (first_transfert)
                {
                        if (sc_sq_data.SC_SQ_state_id == 0)
                                idle0 += 4;
                        else
                                idle1_7 += 4;
                }
                first_transfert = false;

                // get the first pixel group signal and save it
                if (sc_sq_data.SC_SQ_new_vector != 0)
                {
                        interp[SQ_buf_id].new_vector = sc_sq_data.SC_SQ_new_vector;
                        pulsed = true;
                        pSQ_SC->SetSQ_SC_dec_cntr_cnt(true);
                }
                if (sc_sq_data.SC_SQ_pc_dealloc > 0)
                {
                        interp[SQ_buf_id].pc_dealloc += sc_sq_data.SC_SQ_pc_dealloc;
                }

                // load the control data in the control buffers
                for (i=0;i<4;i++)
                {
                        if (sc_sq_data.SC_SQ_quad_mask[i])
                        {
                                // get the associated state and save it
                                interp[SQ_buf_id].state_id = sc_sq_data.SC_SQ_state_id;

                                interp[SQ_buf_id].noIncrement                   =
sc_sq_data.SC_SQ_no_inc_pix_cnt;
                                interp[SQ_buf_id].ptr0[lineSQ[i]%4][i]          =
sc_sq_data.SC_SQ_pc_ptr0;
                                interp[SQ_buf_id].ptr1[lineSQ[i]%4][i]          =
sc_sq_data.SC_SQ_pc_ptr1;
                                interp[SQ_buf_id].ptr2[lineSQ[i]%4][i]          =
sc_sq_data.SC_SQ_pc_ptr2;
                                interp[SQ_buf_id].provok[lineSQ[i]%4][i]        =
sc_sq_data.SC_SQ_provok_vtx;
```

```
                                        interp[SQ_buf_id].pix_mask[lineSQ[i]%4][i]              =
sc_sq_data.SC_SQ_pix_mask[i];
                                        interp[SQ_buf_id].lod_correct[lineSQ[i]%4][i]           =
sc_sq_data.SC_SQ_lod_correct[i].getValue();

                                        // get the primitive type
                                        interp[SQ_buf_id].prim_type[lineSQ[i]%4][i]             =
sc_sq_data.SC_SQ_prim_type;

                                        lineSQ[i] = (lineSQ[i]+1)%4;
                                        QWrote ++;
                                }
                        }

                        // manage completion of a pixel vector
                        if (QWrote == 16)
                        {
                                QWrote = 0;

                                // a valid non event vector is 100
                                pix.event = 100;
                                eventFIFO.push(pix);
                                first_transfert = true;

                                setContextNumber(interp[SQ_buf_id].state_id.getValue());
                                // increment by one more buffer is sending two buffers down
                                if      (SQ_CONTEXT_MISC.getSC_SAMPLE_CNTL()      ==
CENTROIDS_AND_CENTERS){
                                        SQ_buf_id = (SQ_buf_id+1)%4;
                                }
                                SQ_buf_id = (SQ_buf_id+1)%4;

                        }
                }

                // if the event fifo contains something, try to put it in the RS
                if (!eventFIFO.empty())
                {
                        pix = eventFIFO.front();
                        if (pix.event < 100)
                        {
                                if (pix.event == RST_PIX_CNT)
                                {
                                        if (pix.state == 0)
                                                idle0 --;
                                        else
```

```
                                        idle1_7 --;
                                auto_count_pix = 0;
                                eventFIFO.pop();
                        }
                        else if (!arbiter->AddVector(pix.event,PIXEL,pix.state,interp[buf_read].pix_mask,true,interp[buf_read].lod_correct))
                        {
                                eventFIFO.pop();
                        }
                }
                else if (pix.event == 100 && !pulsed)
                {
                        ready=1;
                }
                else if (pix.event >= 200 && pix.event < 300)
                {
                        deallocating = pix.event - 200;
                        eventFIFO.pop();
                        if (pix.state == 0)
                                idle0 --;
                        else
                                idle1_7 --;
                }
                // new vector
                else if (pix.event == 300)
                {
                        if (vertexReady[pix.state]>0)
                        {
                                vertexReady[pix.state]--;
                                eventFIFO.pop();
                                if (pix.state == 0)
                                        idle0 --;
                                else
                                        idle1_7 --;
                        }
                }
        }

        // accumulate data interface
        if (sc_sp_data.SC_SP_valid)
        {
                for (i=0;i<4;i++)
                {
                        if (sc_sp_data.SC_SP_valid[i])
                        {
```

```
                                        // ij data
                                        if (sc_sp_data.SC_SP_type[i] == CENTROID)
                                        {
                                                for (j=0;j<4;j++)
                                                {
                                                        interp[lineSP[i]/4].I[lineSP[i]%4][i*4+j]    =
sc_sp_data.SC_SP_ij_data[i].I[j];
                                                        interp[lineSP[i]/4].J[lineSP[i]%4][i*4+j]    =
sc_sp_data.SC_SP_ij_data[i].J[j];
                                                }
                                        }
                                        else if (sc_sp_data.SC_SP_type[i] == CENTER)
                                        {
                                                for (j=0;j<4;j++)
                                                {

        interp[(lineSP[i]/4+1)%4].I[lineSP[i]%4][i*4+j] = sc_sp_data.SC_SP_ij_data[i].I[j];

        interp[(lineSP[i]/4+1)%4].J[lineSP[i]%4][i*4+j] = sc_sp_data.SC_SP_ij_data[i].J[j];
                                                }
                                        }
                                        // xy data
                                        else if (sc_sp_data.SC_SP_type[i] == XY_FACENESS)
                                        {
                                                interp[lineSP[i]/4].X[lineSP[i]%4][i]                =
(sc_sp_data.SC_SP_ij_data[i].I[0] >> 12) & 0xfff;
                                                interp[lineSP[i]/4].Y[lineSP[i]%4][i]                =
(sc_sp_data.SC_SP_ij_data[i].I[0] & 0xfff);
                                                interp[lineSP[i]/4].face[lineSP[i]%4][i]             =
(sc_sp_data.SC_SP_ij_data[i].I[0] >> 24) & 0x1;
                                        }

                                        // change line in the SP
                                        if (sc_sp_data.SC_SP_last_quad_data[i])
                                        {
                                                // if sending more than one buffer
                                                if ((lineSP[i]+1)%4 == 0)
                                                {

        setContextNumber(interp[lineSP[i]/4].state_id.getValue());
                                                        if
(SQ_CONTEXT_MISC.getSC_SAMPLE_CNTL() == CENTROIDS_AND_CENTERS)
                                                                lineSP[i] = (lineSP[i]+4)%16;
                                                }
                                                lineSP[i] = (lineSP[i]+1)%16;
                                        }
```

```
                        }
                    }
                }

                // if IJ buffer filled, interpolate the results
                // also allocate the GPRs.
                if (ready > 0)
                {
                    // set the state to the current state
                    setContextNumber(interp[buf_read].state_id.getValue());

                    int base_ptr;
                    int numReg;
                    numReg = SQ_PROGRAM_CNTL.getPS_NUM_REG()+1;
                    boolean GPR_full = true;
                    boolean station_full =true;
                    int address;

                    // if the data is ready in the PC
                    if                    (!interp[buf_read].new_vector                    ||
vertexReady[interp[buf_read].state_id]>0 ||
                        interp[buf_read].prim_type[0][0] >= 4) // Real Time
                    {
                        // check for space in both GPRs and reservation station 0
                        GPR_full = gpr_manager->testAllocate(numReg,base_ptr,PIXEL);
                        if (!GPR_full)
                        {
                            station_full = arbiter->AddVector(base_ptr,PIXEL,

        interp[buf_read].state_id,interp[buf_read].pix_mask,false,
                                                                interp[buf_read].lod_correct);
                        }

                        // if we have place for everything AND there is valid data
                        // in the PCs if this is the first vector...
                        if (!GPR_full && !station_full)
                        {
                            // Structure for the SQ->SP dummy interface
                            SQ_SP_interp_data interpData;

                            // clear the firstVector flag and decrement the count if
                            // the pixel group was accepted
                            if (interp[buf_read].new_vector)
                            {
                                interp[buf_read].new_vector = false;
                                vertexReady[interp[buf_read].state_id]--;
```

```
}

gpr_manager->allocate(numReg,base_ptr,PIXEL);
// loop for the four lines
for (j=0;j<4;j++)
{
	address = base_ptr;
	int IJlineIndex;
	// loop for the number of parameters to interpolate
	int interp_params;
	if (SQ_PROGRAM_CNTL.getPARAM_GEN()
&& SQ_PROGRAM_CNTL.getGEN_INDEX_PIX())
		interp_params =
SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+3;
	else if
(SQ_PROGRAM_CNTL.getPARAM_GEN() ||
SQ_PROGRAM_CNTL.getGEN_INDEX_PIX())
		interp_params =
SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+2;
	else
		interp_params =
SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1;

	if (interp_params > 16)
		interp_params = 16;

	for (i=0;i<interp_params;i++)
	{
		int shade =
SQ_INTERPOLATOR_CNTL.getPARAM_SHADE();
		bool flat = !((shade >> (interp_params-
1))&0x01);

		// deal with the center/centroid stuff here
		IJlineIndex = j;
		uint ijBuffer;
		ijBuffer = buf_read;
		if
(SQ_INTERPOLATOR_CNTL.getSAMPLING_PATTERN() != 0)
		{
			int samplingPattern =
SQ_INTERPOLATOR_CNTL.getSAMPLING_PATTERN();
			if ((samplingPattern >> i)&0x01)
				ijBuffer = (buf_read+1)%4;
		}
```

```
interpolators.Interpolate(regFile[j],address,interp[ijBuffer].I[IJlineIndex],
                                                    interp[ijBuffer].J[IJlineIndex],

interp[buf_read].ptr0[j],interp[buf_read].ptr1[j],

interp[buf_read].ptr2[j],i,interp[buf_read].prim_type[j],this,

interp[buf_read].X[j],interp[buf_read].Y[j],interp[buf_read].face[j],flat,interp[buf_read].p
rovok[j],!interp[buf_read].noIncrement);

                                                    // write to the SP dummy interface
                                                    RegVect* values;

                                                    regFile[j]->GetValues(values,address);

                                                    interpData.Address[i]=i+base_ptr;
                                                    interpData.NumParams = interp_params;

                                                    for (int k=0;k<16;k++)
                                                    {

interpData.InterpData[i][k][j].field[0]=values[k].field[0];

interpData.InterpData[i][k][j].field[1]=values[k].field[1];

interpData.InterpData[i][k][j].field[2]=values[k].field[2];

interpData.InterpData[i][k][j].field[3]=values[k].field[3];
                                                    }
                                                    // increment the GPR address
                                                    if (address+1 < gpr_manager->pixLimit)
                                                    {
                                                            address ++;
                                                    }
                                                    else
                                                    {
                                                            address = 0;
                                                    }
                                            }
                                    }
                                    pSQ_SP_Interp->SetAll(&interpData);
                                    pSQ_SP_Interp->SetValid(true);

                                    // dump the SQ->SP interpolator dummy interface
                                    if(m_dumpSQ>0) {
```

```
                                        if (pSQ_SP_Interp->GetNewValid())
                                        {
                                          pSQ_SP_Interp-
>GetNewAll(&(m_sqSpInterpDump->_data));
                                          m_sqSpInterpDump->Dump();
                                        }
                                }

                                // signify to the SC that we freed a buffer
                                pSQ_SC->SetSQ_SC_free_buf(true);
                                // And a control line
                                pSQ_SC->SetSQ_SC_dec_cntr_cnt(true);

                                // pop the event queue to signify that we consumed a buffer
                                eventFIFO.pop();

                                // set the deallocation flags
                                if (interp[buf_read].pc_dealloc >0)
                                {
                                        deallocating = interp[buf_read].pc_dealloc;
                                        interp[buf_read].pc_dealloc = 0;
                                }

                                // swap buffers
                                buf_read = (buf_read+1)%4;
                                // increment one more if multiple buffers for current state
                                if   (SQ_CONTEXT_MISC.getSC_SAMPLE_CNTL()   ==
CENTROIDS_AND_CENTERS)
                                        buf_read = (buf_read+1)%4;

                        } // endif GPR and RS ready
                } // endif data ready
        } // endif ready process pixel

        // dump the SQ->SC interface
        if(m_dumpSQ>0) {
                if (pSQ_SC->GetNewValid())
                {
                  pSQ_SC->GetNewAll(&(m_sqScDump->_data));
                  m_sqScDump->Dump();
                }
        }

        //PC Deallocation
        static int deallocation = 0;
        int dealloc;
```

```
		while (deallocating > 0)
		{
			// new dealocation scheme (groups of 16)
			if (pcAllocated.empty())
			{
				std::cerr << "Error in SQ, trying to dealocate empty parameter
stores" << std::endl;
			}
			dealloc = pcAllocated.front()/4;
			deallocation ++;

			if (deallocation == 4)
			{
				pcAllocated.pop();
				deallocation = 0;
			}

			if (pcFree + dealloc < 128)
				pcFree += dealloc;
			else
			{
				pcFree = dealloc-(128-pcFree);
				checkHigh = !checkHigh;
			}
			deallocating --;
		} // end while PC dealloc
	}

	void cUSER_BLOCK_SQ::ProcessVerts(void)
	{
		static int stageCount = 0;

		// current staging register layer
		static int layer =0;

		static bool doubleSent = false;

		static uinteger<4> valids[4][4];

		static bool ready = false;

		// used to keep the state around if we need to stall
		static uinteger<3> vState;

		// compute the number of valid pipes
```

```
		int dis = pChip->pROM->ROM_BAD_PIPE_DISABLE_REGISTER.DISABLE_SP_VTX;

		if (vgt_sq_verts_data.VGT_SQ_send && !ready && !vgt_sq_verts_data.VGT_SQ_event)
		{
			vState = vgt_sq_verts_data.VGT_SQ_state;

			RegVect value;
			value.field[0]= vgt_sq_verts_data.VGT_SQ_vsisr_data[0];
			value.field[1]= vgt_sq_verts_data.VGT_SQ_vsisr_data[1];
			value.field[2]= vgt_sq_verts_data.VGT_SQ_vsisr_data[2];

			stagingRegisters[stageCount][layer] = value;

			if (stageCount == 0 && layer == 0)
			{
				if (vState == 0)
					idle0 += 4;
				else
					idle1_7 += 4;
			}

			if ((stageCount%4) == 0 && layer==0)
				valids[stageCount/16][(stageCount/4)%4] =0;

			// only validate if VsisrData is valid
			if (vgt_sq_verts_data.VGT_SQ_indx_valid)
			{
				if (layer == 0)
					valids[stageCount/16][(stageCount/4)%4] += 1<<(stageCount%4);
				stageCount++;
				if (stageCount%4 == 0)
				{
					if (((stageCount == 16 || stageCount == 32 || stageCount == 48) && dis&0x01) ||
						((stageCount == 4  || stageCount == 20 || stageCount == 36 || stageCount == 52) && dis&0x02) ||
						((stageCount == 8  || stageCount == 24 || stageCount == 40 || stageCount == 56) && dis&0x04) ||
						((stageCount == 12 || stageCount == 28 || stageCount == 44 || stageCount == 60) && dis&0x08))
					{
						stageCount += 4;
					}
```

```
                }
        }

        // reset the layer to 0
        layer = 0;

        if (vgt_sq_verts_data.VGT_SQ_end_of_vtx_vect)
        {
                for (int i=stageCount;i<64;i++)
                {
                        if ((i%4) == 0)
                                valids[i/16][(i/4)%4] =0;
                }
                if (!vgt_sq_verts_data.VGT_SQ_vsisr_continued)
                        ready = true;
        }

        if (vgt_sq_verts_data.VGT_SQ_vsisr_continued)
        {
                layer = 1;
                if ((stageCount-4)%4 == 0 && (stageCount-4) >0)
                {
                        if (((stageCount == 16 || stageCount == 32 || stageCount ==
48) && dis&0x01) ||
                                ((stageCount == 4  || stageCount == 20 || stageCount
== 36 || stageCount == 52) && dis&0x02) ||
                                ((stageCount == 8  || stageCount == 24 || stageCount
== 40 || stageCount == 56) && dis&0x04) ||
                                ((stageCount == 12 || stageCount == 28 ||
stageCount == 44 || stageCount == 60) && dis&0x08))
                                {
                                        stageCount -= 4;
                                }
                        }
                        stageCount --;
                        doubleSent = true;
                }

                // regular end of vector (not early terminated)
                if (stageCount == 64)
                        ready = true;

        }

        // event processing
        static int eventId;
```

```
		static bool sentEvt = false;
		float tempId;
		static int evState;
		if (vgt_sq_verts_data.VGT_SQ_send && vgt_sq_verts_data.VGT_SQ_event &&
!sentEvt)
		{
			tempId = vgt_sq_verts_data.VGT_SQ_vsisr_data[0].getReal();
			eventId = reinterpret_cast<uint32&>(tempId);
			eventId = eventId & 0x1F;
			// filter out all events but fot the VS_DEALLOC and VS_TS_DEALLOC
			if (eventId == VS_DEALLOC || eventId == VS_DONE_TS // cp events
				|| eventId == CONTEXT_DONE || eventId ==
CACHE_FLUSH_TS
				|| eventId == CACHE_FLUSH || eventId ==
CACHE_FLUSH_AND_INV_TS_EVENT
				|| eventId == CACHE_FLUSH_AND_INV_EVENT)  // Rb events
			{
				sentEvt = true;
				evState = vgt_sq_verts_data.VGT_SQ_state;
				if (evState == 0)
					idle0 ++;
				else
					idle1_7 ++;
			}
			else if (eventId == RST_VTX_CNT)
				auto_count_vtx = 0;
		}

		if (sentEvt)
		{
			if (!arbiter-
>AddVector(eventId,VERTEX,evState,valids,true,interp[0].lod_correct))
			{
				sentEvt = false;
			}
			else // we are full stop sending data
			{
				vgt_sq_verts_data.VGT_SQ_send = false;
				pVGT_SQ_verts_ready->SetReady(false);
			}
		}

		if (ready)
		{
			// set the state to the current vector
			setContextNumber(vState.getValue());
```

```
// copy everything to GPRs
int base_ptr;
int numReg;
numReg = SQ_PROGRAM_CNTL.getVS_NUM_REG()+1;
boolean GPR_full=true;
boolean station_full=true;

// check for space in both GPRs and reservation station 0
GPR_full = gpr_manager->testAllocate(numReg,base_ptr,VERTEX);
if (!GPR_full)
{
        station_full = arbiter->AddVector(base_ptr,VERTEX,
                                vState,valids,false,interp[0].lod_correct);
}

if (!GPR_full && !station_full)
{
        gpr_manager->allocate(numReg,base_ptr,VERTEX);
        // allocation succesfull write the data
        int i,j;
        RegVect* vtAddr;
        RegVect* vtAddr1;
        RegVect* vtAuto;
        int address;

        for (j=0;j<4;j++)
        {
                // counting GPRs in reverse order for vertices
                address = base_ptr;
                regFile[j]->GetValues(vtAddr,address);
                if (address > gpr_manager->vertLimit)
                        address --;
                else
                        address = 127;
                regFile[j]->GetValues(vtAddr1,address);
                if (address > gpr_manager->vertLimit)
                        address --;
                else
                        address = 127;
                regFile[j]->GetValues(vtAuto,address);
                for (i=0;i<16;i++)
                {
                        vtAddr[i]=stagingRegisters[j*16+i][0];
                        if (doubleSent)
                        {
```

```
                                        vtAddr1[i]=stagingRegisters[j*16+i][1];
                                    }
                                    // auto generated index
                                    if
(SQ_PROGRAM_CNTL.getGEN_INDEX_VTX())
                                    {
                                        vtAuto[i].field[0]=auto_count_vtx;
                                        auto_count_vtx ++;
                                    }
                                }
                            }

                            // clear the buffers
                            stageCount = (dis&0x01)*4;
                            layer = 0;
                            doubleSent = false;
                            ready = false;

                        }
                        else // we are full
                        {
                            vgt_sq_verts_data.VGT_SQ_send = false;
                            pVGT_SQ_verts_ready->SetReady(false);
                        }
                    }
                }

        //*****************************************************************************
*******
        // Process function for block
        //*****************************************************************************
*******
        void cUSER_BLOCK_SQ:: Process(void)
        {
            ProcessVerts();
            ProcessPixels();

            // execute the arbiter
            arbiter->Execute();
        }

        //*****************************************************************************
*******
        // Output function for block
        //*****************************************************************************
*******
```

```
void cUSER_BLOCK_SQ::Output(void)
{
	int i;
	static int current_export = 0;
	static int export_count = 0;
	static int currentPtr[4];

	if (outBuffer.valid)
	{
		outBuffer.valid = false;
		// VERTEX  PARAMETER CACHE EXPORT
		if ((outputType == VERTEX) && (currentExportDest < 16))
		{
			int pcPointer;
			// new export block reset the counts
			currentPtr[0] = currentAluPC;
			currentPtr[1] = (currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1))%128;
			currentPtr[2] = (currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1)*2)%128;
			currentPtr[3] = (currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1)*3)%128;

			// set pcPointer to the correct value
			pcPointer = (currentPtr[current_export] + currentExportDest)%128;

			// copy data to the PCs
			int valid;
			for (i=0;i<16;i++)
			{
				valid = outBuffer.valids[i/4].getValue();
				if ((valid >> i%4) &0x01)
				{
					if (export_mask & 0x01)
						parameters[pcPointer].Val[i].field[0] = outBuffer.values[i].field[0];
					if (export_mask & 0x02)
						parameters[pcPointer].Val[i].field[1] = outBuffer.values[i].field[1];
					if (export_mask & 0x04)
						parameters[pcPointer].Val[i].field[2] = outBuffer.values[i].field[2];
					if (export_mask & 0x08)
						parameters[pcPointer].Val[i].field[3] = outBuffer.values[i].field[3];
```

```
                        }
                    }

                    // dump the values to a file
                    if(m_dumpSQ>0) {
                        dumpPcValues(export_mask, pcPointer, outBuffer);
                    }

                    current_export++;
                    if (current_export == 4)
                    {
                        current_export=0;
                    }
                }// end parameter cache export
                // other exports
                else
                {
                    pSP_SX->SetValid(true);
                    for (i=0;i<16;i++)
                    {
                        pSP_SX-
>SetSP_SX_color(outBuffer.values[i].field[0],i*4);
                        pSP_SX-
>SetSP_SX_color(outBuffer.values[i].field[1],i*4+1);
                        pSP_SX-
>SetSP_SX_color(outBuffer.values[i].field[2],i*4+2);
                        pSP_SX-
>SetSP_SX_color(outBuffer.values[i].field[3],i*4+3);
                        pSP_SX->SetSP_SX_exp_pvalid(outBuffer.valids[i/4],i/4);
                    }
                    uinteger<6> dest;
                    dest = currentExportDest;

                    pSP_SX->SetSP_SX_dest(dest);
                    pSP_SX->SetSP_SX_alu_id(currentExportAlu);
                    uinteger<2> exp_count;
                    exp_count = export_count;
                    pSP_SX->SetSP_SX_export_count(exp_count);
                    export_count = (export_count+1)%4;

                    pSP_SX->SetType(outputType);

                    if(m_dumpSQ>0) {
                      pSP_SX->GetNewAll(&(m_spSxDump->_data));
                      m_spSxDump->Dump();
                    }
```

```
                } // end other exports
            }
        }

        bool cUSER_BLOCK_SQ::handleRegisterAccess(ACCESS access, uint32 spaceOffset,
uint32 byteEnable, uint32& data)
        {
           bool handled = false;
                static int count = 0;
                TConstPacked tstate;
                Loop loop;
                uint32 cfBool;
                uint32 gfxDecode;

           if (access == WRITE_ACCESS)
           {
                        // Remove GFX decode from spaceOffset
              if (spaceOffset >= 0x8000 && spaceOffset < 0x10000)
              {
                 gfxDecode = (spaceOffset >> 12) & 0x7;
                 spaceOffset = spaceOffset & ~(0x7 << 12);
              }

                        // grab the CP_STATE_COPY
                        if (spaceOffset == (mmGFX_COPY_STATE<<2))
                        {
                                int previous_write_state = data & 0x7;
                                current_write_state = gfxDecode;

                                // clear the vertex ready counts for the new state to come (may
have been screwed up
                                // by the mem exports.
                                vertexReady[current_write_state]=0;

                                // copy the constant tables
                                int i;
                                for (i=0;i<512;i++)
                                {
                                        constantStore[previous_write_state].GetConstValue(cst,i);
                                        constantStore[current_write_state].WriteValue(cst,i);
                                }
                                for (i=0;i<32;i++)
                                {

        textureStateStore[previous_write_state].GetConstTState(tstate,i);
                                        textureStateStore[current_write_state].WriteTState(tstate,i);
```

```
                        }
                        for (i=0;i<8;i++)
                        {
                                cfBool                                                  =
controlFlowStore[previous_write_state].GetConstBooleans(i);

        controlFlowStore[current_write_state].WriteBooleans(cfBool,i);
                        }
                        for (i=0;i<32;i++)
                        {

        controlFlowStore[previous_write_state].GetConstLoop(loop,i);
                                controlFlowStore[current_write_state].WriteLoop(loop,i);
                        }
                }
            else if ((spaceOffset >= (mmSQ_INSTRUCTION_ALU_0<<2)) && (spaceOffset <
((mmSQ_INSTRUCTION_ALU_0 + 4096*3)<<2)))
            {
                        int         address         =         ((spaceOffset>>2)         -
(mmSQ_INSTRUCTION_ALU_0)) /3;
                        Packet pckt;
                        pckt = reinterpret_cast<Packet&>(data);
                        switch (count){
                        case 0:
                                inst.byte0 = pckt.byte0;
                                inst.byte1 = pckt.byte1;
                                inst.byte2 = pckt.byte2;
                                inst.byte3 = pckt.byte3;
                                break;
                        case 1:
                                inst.byte4 = pckt.byte0;
                                inst.byte5 = pckt.byte1;
                                inst.byte6 = pckt.byte2;
                                inst.byte7 = pckt.byte3;
                                break;
                        case 2:
                                inst.byte8 = pckt.byte0;
                                inst.byte9 = pckt.byte1;
                                inst.byte10 = pckt.byte2;
                                inst.byte11 = pckt.byte3;
                                break;
                        };
                        count ++;

                        // write the instruction to instruction memory
                        if (count == 3)
```

```
                        {
                                count = 0;
                                instructionStore.SetInst(inst,address);
                        }

            handled = true;
          }
          else if ((spaceOffset >= (mmSQ_CONSTANT_RT_0<<2)) && (spaceOffset <
((mmSQ_CONSTANT_RT_0 + 256*4)<<2)))
          {
            int address = ((spaceOffset>>2) - (mmSQ_CONSTANT_RT_0)) /4;
                        cst.field[count].putField(data);
                        count ++;
                        if (count == 4)
                        {
                                count = 0;
                                constantStore[0].WriteValue(cst,address);
                        }

            handled = true;
          }
          else if ((spaceOffset >= (mmSQ_CONSTANT_0<<2)) && (spaceOffset <
((mmSQ_CONSTANT_0 + 512*4)<<2)))
          {
            int address = ((spaceOffset>>2) - (mmSQ_CONSTANT_0)) /4;
                        cst.field[count].putField(data);
                        count ++;
                        if (count == 4)
                        {
                                count = 0;

        constantStore[current_write_state].WriteValue(cst,address);
                        }

            handled = true;
          }
                  else if ((spaceOffset >= (mmSQ_FETCH_0<<2)) && (spaceOffset <
((mmSQ_FETCH_0 + 32*6)<<2)))
          {
            int address = ((spaceOffset>>2) - (mmSQ_FETCH_0)) / 6;
                        tStateData[count] = data;
                        count ++;
                        if (count == 6)
                        {
                                count = 0;
                                tstate.unpack(tStateData);
```

```
        textureStateStore[current_write_state].WriteTState(tstate,address);
                        }
            handled = true;
        }
                else if ((spaceOffset >= (mmSQ_FETCH_RT_0<<2)) && (spaceOffset <
((mmSQ_FETCH_RT_0 + 32*6)<<2)))
        {
            int address = ((spaceOffset>>2) - (mmSQ_FETCH_RT_0)) / 6;
                        tStateData[count] = data;
                        count ++;
                        if (count == 6)
                        {
                                count = 0;
                                TConstPacked tstate;
                                tstate.unpack(tStateData);
                                textureStateStore[0].WriteTState(tstate,address);
                        }
            handled = true;
        }
                else if ((spaceOffset >= (mmSQ_CF_BOOLEANS<<2)) && (spaceOffset
< ((mmSQ_CF_BOOLEANS + 8)<<2)))
        {
                        int address = ((spaceOffset>>2) - (mmSQ_CF_BOOLEANS));


        controlFlowStore[current_write_state].WriteBooleans(data,address);
            handled = true;
        }
                else if ((spaceOffset >= (mmSQ_CF_LOOP<<2)) && (spaceOffset <
((mmSQ_CF_LOOP + 32)<<2)))
        {
                        int address = ((spaceOffset>>2) - (mmSQ_CF_LOOP));
                        Loop loop;

                        loop.count = data & 0xFF;
                        loop.start= (data >> 8) & 0xFF;
                        loop.step = (data >> 16) & 0xFF;

                        controlFlowStore[current_write_state].WriteLoop(loop,address);
            handled = true;
        }
                else if ((spaceOffset >= (mmSQ_CF_RT_BOOLEANS<<2)) &&
(spaceOffset < ((mmSQ_CF_RT_BOOLEANS + 8)<<2)))
        {
                        int address = ((spaceOffset>>2) - (mmSQ_CF_RT_BOOLEANS));
```

```
                        controlFlowStore[0].WriteBooleans(data,address);
                handled = true;
            }
                    else if ((spaceOffset >= (mmSQ_CF_RT_LOOP<<2)) && (spaceOffset <
((mmSQ_CF_RT_LOOP + 32)<<2)))
            {
                        int address = ((spaceOffset>>2) - (mmSQ_CF_RT_LOOP));
                        Loop loop;

                        loop.count = data & 0xFF;
                        loop.start= (data >> 8) & 0xFF;
                        loop.step = (data >> 16) & 0xFF;

                        controlFlowStore[0].WriteLoop(loop,address);
                handled = true;
            }
                    else if ((spaceOffset >= (mmSQ_RT_V0_PARAM0_R<<2)) &&
(spaceOffset < ((mmSQ_RT_V0_PARAM0_R + 16*3*4)<<2)))
                    {
                        int address = ((spaceOffset>>2) - (mmSQ_RT_V0_PARAM0_R));

        RTParameters[address/(16*4)][address/4].field[address%4]==reinterpret_cast<float&>(d
ata);
                        handled = true;
                    }
        }

        return handled;
    }

    void cUSER_BLOCK_SQ::setParameter(float param, int index, int memNum, int field)
    {
            parameters[index].Val[memNum].field[field]=param;
    }

    bool cUSER_BLOCK_SQ::Idle()
    {
            bool idle=true;

            if (idle0 > 0 || idle1_7 >0)
                    idle = false;

    #ifdef DEBUG_SEQ
            static bool prev_idle = true;
            if (idle != prev_idle)
```

```
		{
			if (idle)
				std::cerr << "Sequencer Idle" << std::endl;
			else
				std::cerr << "Sequencer Active" << std::endl;
			prev_idle = idle;
		}
	#endif

		return idle;
	}

	bool cUSER_BLOCK_SQ::Idle0()
	{
		if (idle0 >0)
			return false;
		else
			return true;
	}

	bool cUSER_BLOCK_SQ::Idle1_7()
	{
		if (idle1_7 >0)
			return false;
		else
			return true;
	}

	void cUSER_BLOCK_SQ::dumpPcValues(int expmask,int pcPointer,const OutBuffer&
values)
	{
		static bool first = true;
		int i;

		if (first)
		{
			first = false;
			fprintf(pcFile,"--PC Pointer     (PC)   ( 7 bits)\n");
			fprintf(pcFile,"--Channel Mask   (MSK)  ( 4 bits)\n");
			fprintf(pcFile,"--Data    Mask   (VAL)  ( 16 bits)\n");
			fprintf(pcFile,"--Colors             (COL)  ( 32 bits)\n");
			fprintf(pcFile,"--P  M V    C          C          C          C          C
C         C         C         C         C         C         C         C         C         C
C         C         C         C         C         C         C         C         C         C
C         C         C         C         C         C         C         C         C         C
C         C         C         C         C         C         C         C         C         C
```

```
C       C       C       C       C       C       C       C       C       C
C       C       C       C       C       C       C       C       C \n");
                fprintf(pcFile,"--C  S A   0       0       0       0       0       0
0       0       0       0       1       1       1       1       1       1       1
1       1       1       2       2       2       2       2       2       2       2
2       2       3       3       3       3       3       3       3       3       3
3       4       4       4       4       4       4       4       4       4       4
5       5       5       5       5       5       5       5       5       5       6
6       6       6 \n");
                fprintf(pcFile,"--   K L   0       1       2       3       4       5
6       7       8       9       0       1       2       3       4       5       6
7       8       9       0       1       2       3       4       5       6       7
8       9       0       1       2       3       4       5       6       7       8
9       0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9       0
1       2       3 \n");
        }

        fprintf(pcFile,"                %02x        %x          %x%x%x%x%x",
pcPointer,expmask,outBuffer.valids[0].getValue(),

    outBuffer.valids[1].getValue(),outBuffer.valids[2].getValue(),outBuffer.valids[3].getVal
ue());

        for (i=0;i<16;i++)
        {
            fprintf(pcFile,"% 010.5e % 010.5e % 010.5e % 010.5e ",
            outBuffer.values[i].field[0].getReal(),
            outBuffer.values[i].field[1].getReal(),
            outBuffer.values[i].field[2].getReal(),
            outBuffer.values[i].field[3].getReal());
        }

        fprintf(pcFile,"\n");

        float var[4];
        fprintf(pcFile,"        ");

        for (i=0;i<16;i++)
        {
            for (int w=0;w<4;w++)
                var[w] = outBuffer.values[i].field[w].getReal();

            fprintf(pcFile,"%08x    %08x    %08x    %08x    ",
                        *(reinterpret_cast<unsigned int*>(&var[0])),
                        *(reinterpret_cast<unsigned int*>(&var[1])),
```

```
                                        *(reinterpret_cast<unsigned int*>(&var[2])),
                                        *(reinterpret_cast<unsigned int*>(&var[3]))
                                        );
                }

                fprintf(pcFile,"\n");
        }
```

Sq_block_model.cpp

```
//************************************************************************
// Output function for block
//************************************************************************
void cUSER_BLOCK_SQ::Output(void)
{
        int i;
        static int current_export = 0;
        static int export_count = 0;
        static int currentPtr[4];

        if (outBuffer.valid)
        {
                outBuffer.valid = false;
                // VERTEX  PARAMETER CACHE EXPORT
                if ((outputType == VERTEX) && (currentExportDest < 16))
                {
                        int pcPointer;
                        // new export block reset the counts
                        currentPtr[0] = currentAluPC;
                        currentPtr[1] =
(currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1))%128;
                        currentPtr[2] =
(currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1)*2)%128;
                        currentPtr[3] =
(currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1)*3)%128;

                        // set pcPointer to the correct value
                        pcPointer = (currentPtr[current_export] + currentExportDest)%128;

                        // copy data to the PCs
                        int valid;
                        for (i=0;i<16;i++)
                        {
                                valid = outBuffer.valids[i/4].getValue();
                                if ((valid >> i%4) &0x01)
                                {
                                        if (export_mask & 0x01)
                                                parameters[pcPointer].Val[i].field[0] =
outBuffer.values[i].field[0];
                                        if (export_mask & 0x02)
                                                parameters[pcPointer].Val[i].field[1] =
outBuffer.values[i].field[1];
                                        if (export_mask & 0x04)
                                                parameters[pcPointer].Val[i].field[2] =
outBuffer.values[i].field[2];
                                        if (export_mask & 0x08)
```

```
                                        parameters[pcPointer].Val[i].field[3] =
outBuffer.values[i].field[3];
                                }
                        }

                        // dump the values to a file
                        if(m_dumpSQ>0) {
                                dumpPcValues(export_mask, pcPointer, outBuffer);
                        }

                        current_export++;
                        if (current_export == 4)
                        {
                                current_export=0;
                        }
                }// end parameter cache export
                // other exports
                else
                {
                        pSP_SX->SetValid(true);
                        for (i=0;i<16;i++)
                        {
                                pSP_SX->SetSP_SX_color(outBuffer.values[i].field[0],i*4);
                                pSP_SX->SetSP_SX_color(outBuffer.values[i].field[1],i*4+1);
                                pSP_SX->SetSP_SX_color(outBuffer.values[i].field[2],i*4+2);
                                pSP_SX->SetSP_SX_color(outBuffer.values[i].field[3],i*4+3);
                                pSP_SX->SetSP_SX_exp_pvalid(outBuffer.valids[i/4],i/4);
                        }
                        uinteger<6> dest;
                        dest = currentExportDest;

                        pSP_SX->SetSP_SX_dest(dest);
                        pSP_SX->SetSP_SX_alu_id(currentExportAlu);
                        uinteger<2> exp_count;
                        exp_count = export_count;
                        pSP_SX->SetSP_SX_export_count(exp_count);
                        export_count = (export_count+1)%4;

                        pSP_SX->SetType(outputType);

                        if(m_dumpSQ>0) {
                           pSP_SX->GetNewAll(&(m_spSxDump->_data));
                           m_spSxDump->Dump();
                        }
                } // end other exports
```

Regarding fetching data from memory, The texture fetcher allows fetching from memory. The arbiter.cpp file picks the programs that need to fetch data in this function:

```
boolean Arbiter::chooseTexStation(int &lineNumber,Shader_Type &sType)
{
        int i;
        int vertexPick = -1;
        int pixelPick = -1;
        int lineCheck;

        // do pixels first
```

```
	lineCheck = pixelHead;
	for (i=0;i<pixelRsCount;i++)
	{
		if (pixelStation[lineCheck].status.valid &&
pixelStation[lineCheck].status.ressourceNeeded == TEXTURE
			&& !pixelStation[lineCheck].status.event)
		{
			pixelPick=lineCheck;
		}
		// enforce restrictions based on the status
		if (pixelPick != -1)
		{
			// no texture ops while texture reads are outstanding
			if (pixelStation[pixelPick].status.texReadsOutstanding)
				pixelPick = -1;
			else
				break;
		}

		lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
	}

	lineCheck = vertexHead;
	for (i=0;i<vertexRsCount;i++)
	{
		if (vertexStation[lineCheck].status.valid &&
vertexStation[lineCheck].status.ressourceNeeded == TEXTURE
			&& !vertexStation[lineCheck].status.event)
		{
			vertexPick=lineCheck;
		}

		// enforce restrictions based on the status
		if (vertexPick != -1)
		{
			// no texture ops while texture reads are outstanding
			if (vertexStation[vertexPick].status.texReadsOutstanding)
				vertexPick = -1;
			else
				break;
		}

		lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
	}

	if (vertexPick != -1)
	{
		lineNumber = vertexPick;
		sType = VERTEX;
		return true;
	}
	if (pixelPick != -1)
	{
		lineNumber = pixelPick;
		sType = PIXEL;
		return true;
	}
```

```
	return false;
}
```

Then fills in a request in this function:

```
void  Arbiter::fillTextureInterface(int textureInstAddr,int texturePhase,boolean last)
{
	const RegVect* txAddr;
	TXAddr address;
	uinteger<7> registerAddress;
	uinteger<7> writeAddress;
	uint8 maxSize;
	int basePtr = textureCFMachine.stationData->data.gprBase;

	sq->pSQ_TP->SetValid(true);

	// Get the instruction
	TInstrPacked inst;

	// set the state to the current running ALU
	sq->setContextNumber(textureCFMachine.stationData->data.state);

	sq->instructionStore.GetInst(inst,textureInstAddr);
	switch (textureCFMachine.sType)
	{
	case PIXEL:
		maxSize = sq->gpr_manager->pixLimit;
		// compute the addresses (read address)
		if ((inst.getSRC_GPR() + basePtr) < maxSize)
			registerAddress = inst.getSRC_GPR() + basePtr;
		else
			registerAddress = inst.getSRC_GPR()-(maxSize-basePtr);
		// write address
		if ((inst.getDST_GPR() + basePtr) < maxSize)
			writeAddress = inst.getDST_GPR() + basePtr;
		else
			writeAddress = inst.getDST_GPR()-(maxSize-basePtr);
		break;
	case VERTEX:
		maxSize = sq->gpr_manager->vertLimit;
		// compute the addresses (read address)
		if (( basePtr - inst.getSRC_GPR()) >= maxSize)
			registerAddress =  basePtr - inst.getSRC_GPR();
		else
			registerAddress = 128-(inst.getSRC_GPR()-(basePtr-maxSize));
		// write address
		if (( basePtr - inst.getDST_GPR()) >= maxSize)
			writeAddress =  basePtr - inst.getDST_GPR();
		else
			writeAddress = 128-(inst.getDST_GPR()-(basePtr-maxSize));
		break;
	};
	sq->regFile[texturePhase]->GetConstValues(txAddr,registerAddress);
	int i;
	for(i=0;i<16;i++)
	{
	 //Do the swizzle for the TP
```

```
        inst.doSrcSwizzle( txAddr[i].field[0], txAddr[i].field[1], txAddr[i].field[2],
txAddr[i].field[3],
                           address.x, address.y, address.z );
              sq->pSQ_TP->SetSP_TP_fetch_addr(address,i);
       }
       for (i=0;i<4;i++)
       {
              uinteger<4> valids;
              valids = textureCFMachine.stationData->data.valids[texturePhase][i];
              // modify the mask to turn on any pixels that are off if not fetch valid
only
              if (!inst.getFETCH_VALID_ONLY())
              {
                     if (valids.getValue() != 0)
                            valids = 0x0F;
              }

              // now modify the mask based on the predicate vector
              if (inst.getPRED_SELECT())
              {
                     bool pred = (inst.getPRED_CONDITION() == 1);
                     if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4])
                     {
                            // kill the pixel
                            valids = valids.getValue() & 0xE;

                     }
                     if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4+1])
                     {
                            // kill the pixel
                            valids = valids.getValue() & 0xD;

                     }
                     if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4+2])
                     {
                            // kill the pixel
                            valids = valids.getValue() & 0xB;

                     }
                     if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4+3])
                     {
                            // kill the pixel
                            valids = valids.getValue() & 0x7;

                     }
              }
              sq->pSQ_TP->SetSQ_TP_pix_mask(valids,i);

              // send the LOD correction bits
              uinteger<9> LODCorrect;
              LODCorrect = textureCFMachine.stationData-
>data.LodCorrect[texturePhase][i];
              sq->pSQ_TP->SetSQ_TP_lod_correct(LODCorrect,i);
       }
```

```
        sq->pSQ_TP->SetSQ_TP_write_gpr_index(writeAddress);
        sq->pSQ_TP->SetSQ_TP_last(last);
        uinteger<6> line;
        line = textureCFMachine.lineNumber;
        sq->pSQ_TP->SetSQ_TP_thread_id(line);
        sq->pSQ_TP->SetSQ_TP_type(textureCFMachine.sType);
        TConstPacked tpConst;
        sq->textureStateStore[textureCFMachine.stationData-
>data.state].GetConstTState(tpConst,inst.getCONST_INDEX());
        sq->pSQ_TP->SetSQ_TP_const(tpConst);
        sq->pSQ_TP->SetSQ_TP_instr(inst);
        uinteger<3> ctxId;
        ctxId = textureCFMachine.stationData->data.state;
        sq->pSQ_TP->SetSQ_TP_ctx_id(ctxId);

        if(sq->m_dumpSQ>0) {
        sq->pSQ_TP->GetNewAll(&(sq->m_sqTpDump->_data));
        sq->m_sqTpDump->Dump();
    }
}
```

sq_gpr_alloc.v

```
  /*
    hers's a description of the basic operation:

    ________          ________          ________          ________
   0 ___1____ <- tail  0 ___1____ <- tail  0 _________     0 ___3____ <- head, tail
   1 _________ <- head  1 ___2____         1 ___2____ <- tail  1 ___2____
   2 _________         2 ___2____          2 ___2____          2 ___2____
   3 _________         3 ___2____          3 ___2____          3 ___2____
   4 _________         4 _________ <- head  4 _________ <- head  4 ___3____
   5 _________         5 _________         5 _________         5 ___3____
   6 _________ <- max   6 _________ <- max  6 _________ <- max   6 _________ <- max
   7 _________       ,  7 _________       ,  7 _________       ,  7 _________       ,

    - initially, head = tail = 0, and max is set to be one more than the maximum allowable location
    - req 1 allocates one location, head is incremented to 1
    - req 2 allocates three locations, head is incremented to 4
    - another request for 3 spaces would not be granted since there are now only two free locations
    - when the space taken by req 1 is dealloc'd, increment tail to 1 (frees up one location)
    - now req 3 allocates three locations, head is incremented to 7, which is greater than max, so it is
      wrapped around by subtracting max (7 - 6 = 1)
    -

    */

…
// - keep track of the free space -

   wire [PTR_WIDTH-1:0] pix_free;   // number of free pixel locations
   wire [PTR_WIDTH-1:0] vtx_free;   //

   assign pix_free = pix_wrapped_q ? pix_tail_q - pix_head_q : pix_max_q - pix_head_q + pix_tail_q;
   assign vtx_free = vtx_wrapped_q ? vtx_head_q - vtx_tail_q : ~vtx_max_q - vtx_tail_q + vtx_head_q;   //
~vtx_max = 127 - vtx_max
…
wire pix_ok_to_alloc = (pix_alloc_space <= pix_free);    // OK to allocate pixel space
   wire pix_alloc = pix_ok_to_alloc & pix_alloc_req;     // signals the start of a pixel alloc operation
   wire pix_dealloc = pix_dealloc_req;           // signals the start of a pixel dealloc operation (always
OK to dealloc?)
   wire pix_head_wraps = (new_pix_head >= pix_max_q);    // new pix_head wraps
   wire pix_tail_wraps = (new_pix_tail >= pix_max_q);    // new pix_tail wraps
…
    wire vtx_ok_to_alloc = (vtx_alloc_space <= vtx_free); // OK to allocate vertex space
   wire vtx_alloc = vtx_ok_to_alloc & vtx_alloc_req;     // signals the start of a vertex alloc operation
   wire vtx_dealloc = vtx_dealloc_req;           // signals the start of a vertex dealloc operation
   wire vtx_head_wraps = (new_vtx_head <= vtx_max_q);    // new vtx_head wraps
   wire vtx_tail_wraps = (new_vtx_tail <= vtx_max_q);    // new vtx_tail wraps

…
case ( ra_current_state )
          IDLE:
          begin
            // - look for any of the four requests
            // - if the request is accepted
            //   - go to the corresponding acknowledge state
            //   - update the base_ptr register on alloc requests

            if ( pix_alloc )
              begin
                ra_next_state = P_ALLOC_ACK;
                next_pix_alloc_ack = HI;

                if ( pix_head_wraps )
```

```
                  begin
                    next_pix_wrapped = HI;
                    next_pix_head = new_pix_head - pix_max_q;
                  end
                else
                  begin
                    next_pix_head = new_pix_head;
                  end

                next_base_ptr = pix_head_q;  // for pixels, the space starts with the current head pointer
              end

            else if ( vtx_alloc )
              begin
                ra_next_state = V_ALLOC_ACK;
                next_vtx_alloc_ack = HI;

                if ( vtx_head_wraps )
                  begin
                    next_vtx_wrapped = HI;
                    next_vtx_head = new_vtx_head + ~vtx_max_q;   // ~vtx_max = 127 - vtx_max
                    //next_base_ptr = new_vtx_head + ~vtx_max + 1;  // for verts, the space starts with
the new head pointer
                  end
                else
                  begin
                    next_vtx_head = new_vtx_head;
                    //next_base_ptr = new_vtx_head + 1;  // for verts, the space starts with the new head
pointer
                  end

                next_base_ptr = next_vtx_head + 1;  // for verts, the space starts with the new head
pointer
                                                    // (coding trick - commented out lines above explain)

              end
```

Sq_alu_instr_seq.v

```
////////////////////////////////////////////////////////////////////////////////
// sq_alu_instr_seq.v
//
// - receives instruction from alu instr queue (AIQ)
// - reads constants (but data goes directly to ais_output mux)
// - sends instruction to SP over 4 cycles (starting on the correct phase)
…
input [1:0] aiq_export_info;     // {exp_id, pulse_sx}
   input [0:0] aiq_last_in_group;        // last instruction flag
   input [0:0] aiq_last_in_shader;       // last instruction flag
   input [0:0] aiq_thread_type;  // 0: pixel, 1: vertex (shows we operate on either pixel or vertex)
   input [2:0] aiq_context_id;   // context_id (from ctl packet)
   input [5:0] aiq_thread_id;    // clause number
…
// - recall that a 0 here means src is a constant (while 1 means src is a gpr)...

   wire ca_fetch = ~aiq_instr[95];
   wire cb_fetch = ~aiq_instr[94];
   wire cc_fetch = ~aiq_instr[93];

   // - instruction bits 63:61 are used as the const addr msb (these bits are decoded and replaced
   //   before entering the AIQ

   wire [8:0] ca_addr = {aiq_instr[63], aiq_instr[87:80]};
   wire [8:0] cb_addr = {aiq_instr[62], aiq_instr[79:72]};
   wire [8:0] cc_addr = {aiq_instr[61], aiq_instr[71:64]};
…
// ----------------------------
   // -- Input Staging Register --
```

```
   // ----------------------------
   // - need to send the vector type and the thread_id back to the thread buffers when
   //   the all the instructions we wanted to run for this thread are done (this will
   //   cause the thread to become valid again)
   // - register this info in from the AIQ on an AIQ pop in order to hold it until the
   //   AIS is done
...
         case (ais_current_state)
           AIS0:
           // - wait until this machine is started by the AIQ read SM
           // - write OSR data into thread buff on new thread (when there was a previous thread...)
           //   - ais_done does updates the thread_buff and clears the alu_instr_pending status bit
           //   - don't assert ais_done yet if the previous instr was a pred set (wait for the pred set
           //     data to arrive from the SP)
           begin
              ais_instr_stall = HI;

              if ( ais_start )
                begin
                   //if ( aiq_new_thread & osr_valid_q & ~osr_pred_set_flag_q ) ais_done = HI;

                   ais_instr_start = HI;
                   ais_instr_stall = LO;
                   ais_next_state = AIS1;
                end
           end

           AIS1: begin ais_next_state = AIS2; end

           AIS2: begin ais_next_state = AIS3; end

           AIS3: begin ais_next_state = AIS4; end

           // ** the AIQ was just popped by the ACS SM, so now must use info saved in ISR ** //

           AIS4: begin ais_next_state = AIS5; end

           AIS5: begin ais_next_state = AIS6; end

           AIS6:
           begin
              // - the pred set data is loaded now from the previous instr, so assert done now
              // - also write new predicate data into predicate register (in ais_output )

              if ( isr_new_thread_q & osr_pred_set_flag_q ) ais_done = HI;

              ld_osr = HI;
              ais_next_state = AIS7;
           end

           AIS7:
           // - pop the thread off the reservation station buffer when the last instr of the shader is
executed
           // - send free_done when pulse_sx is set, or this is the last instruction of a pixel shader
(since this
           //   is when the pixel export is done)

           begin
              if ( isr_last_in_group_q & ~isr_last_in_shader_q ) ais_done = HI;

              if ( isr_pulse_sx_q ) ais_free_done = HI;   // pixel last logic put into pulse_sx generation

              if ( isr_last_in_shader_q ) ais_pop = HI;

              ais_next_state = AIS0;
           end

        endcase
     end
```

```
    // - end ais state machine
```

The ais machine is the “alu instruction sequencer” it executes instructions on either vertices or pixels depending on type. The file sq_instruction_store.v contains the memory with all the instructions to be performed on either PS or VS:

Sq_instruction_store_v

```
// Access to the is (instruction store) is divided into 4 phases:
// 0: texture instruction read
// 1: alu instruction read
// The alu phase alternates between phases for alu0 and alu1.
// 2: CP write (or read for debug)
// 3: control flow instruction read
// The control flow phase is shared for accesses by alu0, alu1 and tex
// controlled by is_sub_phase.


// address mux
always @(/*AUTOSENSE*/addr or data_cnt or i_alu0_addr
         or i_alu0_cf_addr or i_alu1_addr or i_alu1_cf_addr
         or i_is_phase or i_is_sub_phase or i_tex_addr
         or i_tex_cf_addr or q_rbi_addr_in)
  begin
    // default values
    d_addr     = addr;
    d_we       = 1'b0;
    case (i_is_phase)

      TEX_PHASE :
        begin
          d_addr            = i_tex_addr;
        end

      ALU_PHASE :
        begin
          d_addr            = i_is_sub_phase[0] ? i_alu1_addr : i_alu0_addr;
          d_we              = &data_cnt; // data_cnt == 3
        end

      CP_PHASE :
        begin
          d_addr           = q_rbi_addr_in[11:0]; // top bits are zeros by now
        end

      CF_PHASE :
        begin
          case (i_is_sub_phase)
            2'b00 :
              d_addr       = i_alu0_cf_addr;
            2'b10 :
              d_addr       = i_alu1_cf_addr;
            default :
              d_addr       = i_tex_cf_addr;
          endcase // case(i_is_sub_phase)
        end
    endcase // case(i_is_phase)

  end // always @ (...
```

Claim 2:

sp_macc_gpr.v

```
// Filename        : macc_reg.v
// Description     : This module represents the MACC (Multiply and Accumulate) unit plus
//                 : the correnspounding GPR (register file) module.
// Author          : Andi Skende

rfsd2_128x128cm1sw8_core ugpr_mem(.QB(reg_data),
                                  .ADRA_buf(gpr_wr_addr),
                                  .DA_buf(input_gpr),
                                  .WEMA_buf(subword_write_mask),
                                  .WEA_buf(gpr_wr_ena),
                                  .MEA_buf(gpr_wr_ena),
                                  .CLKA(sclk),
                                  .BISTEA(1'b0),
                                  .ADRB_buf(sq_sp_gpr_rd_addr),
                                  .OEB_buf(1'b1),
                                  .MEB_buf(sq_sp_mem_rd_ena),
                                  .CLKB(sclk),
                                  .BISTEB(1'b0),
                                  .AWTB(1'b0)
                                  );
```

This is the instantiation of the GPR memory, 128x128.

Sp_vector.v (shows the instanciation of 4 multiply accumulate modules and 1 scalar module):

```
//-------------------------------------------------------------------------------------------------------------
----
   //Scalar Unit instantiation
   //-------------------------------------------------------------------------------------------------------------
------
   sp_scalar_lut uscalar(
                         .iAG_ME_OPCODE(scalar_opcode),
                         .iAG_ME_IN_A(scalar_input_alpha),
                         .iAG_ME_IN_B(scalar_input_red),
                         .iAG_ME_IN_C(32'b0),
                         .iAG_ME_ABS_A(scalar_input_abs),
                         .iAG_ME_ABS_B(scalar_input_abs),
                         .iAG_ME_ABS_C(scalar_input_abs),
                         .iAG_ME_A_NEGATE(scalar_input_negate),
                         .iAG_ME_B_NEGATE(scalar_input_negate ^ scalar_opcode_sub),
                         .iAG_ME_C_NEGATE(scalar_input_negate),
                         .oME_RESULT(scalar_result),
                         .sclk(sclk)
                         );


   //replicating the scalar_result (32 bits) to all of the four channels of the write back path into GPRs
   //masking is done at the GPRs input
   assign scalar_result_bus = { scalar_result, scalar_result, scalar_result, scalar_result};


   //-------------------------------------------------------------------------------------------------------------
-------------------------------
   //Instantiation of all four MACC units that create a Vector Unit
   //-------------------------------------------------------------------------------------------------------------
-------------------------------
   sp_macc_gpr usp_macc_gpr0(.ovector_output(VectorResult0),
                             .oscalar_input_alpha(scalar_input0_alpha),
                             .oscalar_input_red(scalar_input0_red),
                             .oscalar_input_negate(scalar_input0_negate),
                             .oscalar_input_abs(scalar_input0_abs),
                             .oscalar_opcode(scalar_opcode0),
                             .oreg_data(RegData0),.oexport_dst(sq_sp_exp_dst),

.sq_sp_instruct(sq_sp_instruct),.sq_sp_instruct_start(sq_sp_instruct_start),.sq_sp_stall(sq_sp_stall),
                             .sq_sp_gpr_rd_addr(sq_sp_gpr_rd_addr),
```

```
.sq_sp_gpr_wr_addr(sq_sp_wr_addr),.sq_sp_wr_ena(sq_sp_wr_ena0),.sq_sp_mem_rd_ena(sq_sp_mem_rd_ena),.sq_sp_
mem_wr_ena(sq_sp_mem_wr_ena0),

.sq_sp_gpr_cmask(sq_sp_channel_mask),.sq_sp_pred_override(sq_sp_pred_override),

.sq_sp_gpr_phase_mux(sq_sp_gpr_phase_mux),.iInterpolated(InputData0),.sq_sp_constant(sq_sp_constant),
                              .iscalar_data(scalar_result_bus),.tp_sp_data(tp_sp_data),
                              .tp_sp_gpr_dst(tp_sp_gpr_dst),
.tp_sp_gpr_cmask(tp_sp_gpr_cmask),.tp_sp_data_valid(tp_sp_data_valid[0]),
                              .sclk(sclk), .srst(srst));

   sp_macc_gpr usp_macc_gpr1(.ovector_output(VectorResult1),
                              .oscalar_input_alpha(scalar_input1_alpha),
                              .oscalar_input_red(scalar_input1_red),
                              .oscalar_input_negate(scalar_input1_negate),
                              .oscalar_input_abs(scalar_input1_abs),
                              .oscalar_opcode(scalar_opcode1),

.oreg_data(RegData1),.sq_sp_instruct(q0_instruct),.sq_sp_instruct_start(q0_instruct_start),.sq_sp_stall(q0
_instruct_stall),
                              .sq_sp_gpr_rd_addr(q0_gpr_rd_addr),

.sq_sp_gpr_wr_addr(q0_gpr_wr_addr),.sq_sp_wr_ena(sq_sp_wr_ena1),.sq_sp_mem_rd_ena(q0_gpr_mre),.sq_sp_mem_w
r_ena(sq_sp_mem_wr_ena1),
                              .sq_sp_gpr_cmask(q0_gpr_cmask),.sq_sp_pred_override(q0_pred_override),

.sq_sp_gpr_phase_mux(q0_gpr_phase_mux),.iInterpolated(InputData1),.sq_sp_constant(q0_sq_constant),
                              .iscalar_data(scalar_result_bus),.tp_sp_data(tp_sp_data),
                              .tp_sp_gpr_dst(q0_tp_gpr_dst),
.tp_sp_gpr_cmask(q0_tp_gpr_cmask),.tp_sp_data_valid(tp_sp_data_valid[1]),
                              .sclk(sclk), .srst(srst));

   sp_macc_gpr usp_macc_gpr2(.ovector_output(VectorResult2),
                              .oscalar_input_alpha(scalar_input2_alpha),
                              .oscalar_input_red(scalar_input2_red),
                              .oscalar_input_negate(scalar_input2_negate),
                              .oscalar_input_abs(scalar_input2_abs),
                              .oscalar_opcode(scalar_opcode2),

.oreg_data(RegData2),.sq_sp_instruct(q1_instruct),.sq_sp_instruct_start(q1_instruct_start),.sq_sp_stall(q1
_instruct_stall),
                              .sq_sp_gpr_rd_addr(q1_gpr_rd_addr),

.sq_sp_gpr_wr_addr(q1_gpr_wr_addr),.sq_sp_wr_ena(sq_sp_wr_ena2),.sq_sp_mem_rd_ena(q1_gpr_mre),.sq_sp_mem_w
r_ena(sq_sp_mem_wr_ena2),
                              .sq_sp_gpr_cmask(q1_gpr_cmask),.sq_sp_pred_override(q1_pred_override),

.sq_sp_gpr_phase_mux(q1_gpr_phase_mux),.iInterpolated(InputData2),.sq_sp_constant(q1_sq_constant),
                              .iscalar_data(scalar_result_bus),.tp_sp_data(tp_sp_data),
                              .tp_sp_gpr_dst(q1_tp_gpr_dst),
.tp_sp_gpr_cmask(q1_tp_gpr_cmask),.tp_sp_data_valid(tp_sp_data_valid[2]),
                              .sclk(sclk), .srst(srst));

   sp_macc_gpr usp_macc_gpr3(.ovector_output(VectorResult3),
                              .oscalar_input_alpha(scalar_input3_alpha),
                              .oscalar_input_red(scalar_input3_red),
                              .oscalar_input_negate(scalar_input3_negate),
                              .oscalar_input_abs(scalar_input3_abs),
                              .oscalar_opcode(scalar_opcode3),

.oreg_data(RegData3),.sq_sp_instruct(q2_instruct),.sq_sp_instruct_start(q2_instruct_start),.sq_sp_stall(q2
_instruct_stall),
                              .sq_sp_gpr_rd_addr(q2_gpr_rd_addr),

.sq_sp_gpr_wr_addr(q2_gpr_wr_addr),.sq_sp_wr_ena(sq_sp_wr_ena3),.sq_sp_mem_rd_ena(q2_gpr_mre),.sq_sp_mem_w
r_ena(sq_sp_mem_wr_ena3),
                              .sq_sp_gpr_cmask(q2_gpr_cmask),.sq_sp_pred_override(q2_pred_override),
```

```
.sq_sp_gpr_phase_mux(q2_gpr_phase_mux),.iInterpolated(InputData3),.sq_sp_constant(q2_sq_constant),
                              .iscalar_data(scalar_result_bus),.tp_sp_data(tp_sp_data),.sclk(sclk),
                              .tp_sp_gpr_dst(q2_tp_gpr_dst),
.tp_sp_gpr_cmask(q2_tp_gpr_cmask),.tp_sp_data_valid(tp_sp_data_valid[3]),
                              .srst(srst));
   //-------------------------------------------------------------------------------------------------------
--------------------------------
   //Muxing the gpr vector results into one final vector result conrolled by the phase_mux signal or a
registered version of it
   //-------------------------------------------------------------------------------------------------------
--------------------------------
```

Sq.v

```
   //-------------------------------------------------------------
   // SQ-SP GPR control Interface
   //-------------------------------------------------------------
   output [6:0] SQ_SP_gpr_wr_addr;
   output [0:0] u0_SQ_SP_gpr_wr_en0;
   output [0:0] u0_SQ_SP_gpr_wr_en1;
   output [0:0] u0_SQ_SP_gpr_wr_en2;
   output [0:0] u0_SQ_SP_gpr_wr_en3;
   output [0:0] u1_SQ_SP_gpr_wr_en0;
   output [0:0] u1_SQ_SP_gpr_wr_en1;
   output [0:0] u1_SQ_SP_gpr_wr_en2;
   output [0:0] u1_SQ_SP_gpr_wr_en3;
   output [0:0] u2_SQ_SP_gpr_wr_en0;
   output [0:0] u2_SQ_SP_gpr_wr_en1;
   output [0:0] u2_SQ_SP_gpr_wr_en2;
   output [0:0] u2_SQ_SP_gpr_wr_en3;
   output [0:0] u3_SQ_SP_gpr_wr_en0;
   output [0:0] u3_SQ_SP_gpr_wr_en1;
   output [0:0] u3_SQ_SP_gpr_wr_en2;
   output [0:0] u3_SQ_SP_gpr_wr_en3;
   output [6:0] SQ_SP_gpr_rd_addr;
   output [0:0] SQ_SP_gpr_rd_en;
   output [1:0] SQ_SP_gpr_phase_mux;
   output [3:0] SQ_SP_channel_mask;

   output [1:0]  SQ_SP_gpr_input_mux;
   output [`AUTO_COUNT_SIZE - 1 :0] SQ_SP_auto_count;

  //-------------------------------------------------------------
   // SQ-SP : Instruction interface
   //-------------------------------------------------------------
   output [0:0] SQ_SP_instruct_start;
   output [0:0] SQ_SP_stall;
   output [23:0] SQ_SP_instruct;
   output [127:0] SQ_SP_const;

   output [0:0] SQ_SP_exporting;
   output [0:0] SQ_SP_exp_id;

   output [7:0] u0_SQ_SX_kill_mask;    // valid bits/kill mask
   output [7:0] u1_SQ_SX_kill_mask;

   output [3:0] u0_SQ_SP_pred_override;
   output [3:0] u1_SQ_SP_pred_override;
   output [3:0] u2_SQ_SP_pred_override;
   output [3:0] u3_SQ_SP_pred_override;
```

Sq_export_alloc.v

```
always @( alloc_cmd )
     begin
        casez ( alloc_cmd )
          // - vtx pos alloc
          7'b1_01_0001 : sx_exp_cmd = 5'b10_00_1;
          7'b1_01_0010 : sx_exp_cmd = 5'b10_01_1;

          // - vtx pass thru
          7'b1_11_0100 : sx_exp_cmd = 5'b11_00_1;
          7'b1_11_1000 : sx_exp_cmd = 5'b11_01_1;
          7'b1_11_1100 : sx_exp_cmd = 5'b11_10_1;

          // - pix without z
          7'b0_10_0010 : sx_exp_cmd = 5'b00_00_1;
          7'b0_10_0100 : sx_exp_cmd = 5'b00_01_1;
          7'b0_10_0110 : sx_exp_cmd = 5'b00_10_1;
          7'b0_10_1000 : sx_exp_cmd = 5'b00_11_1;

          // - pix with z
          7'b0_10_0011 : sx_exp_cmd = 5'b01_00_1;
          7'b0_10_0101 : sx_exp_cmd = 5'b01_01_1;
          7'b0_10_0111 : sx_exp_cmd = 5'b01_10_1;
          7'b0_10_1001 : sx_exp_cmd = 5'b01_11_1;

          // - pix pass thru
          7'b0_11_0100 : sx_exp_cmd = 5'b11_00_1;
          7'b0_11_1000 : sx_exp_cmd = 5'b11_01_1;
          7'b0_11_1100 : sx_exp_cmd = 5'b11_10_1;

          default: sx_exp_cmd = 5'bxxxx0;
        endcase
     end
```

Shows the SQ able to execute any types of export commands (position, pass-thru (appearance), pix (color).

An example of a shared resource is the instruction store, accesses to it are controlled by:

sq_ctl_flow_seq.v

```
module sq_ctl_flow_seq
(
   cfs_type_strap,       // 00:alu0, ?1:tex, 10:alu1

   is_phase,             // 00:CF, 01:Tex, 10:ALU, 11:CP
   is_subphase,          // 00:alu0, 01:tex, 10:alu1, 11:tex
   cfs_phase,            // 00:alu0, 01:tex, 10:alu1, 11:tex
   cfc_phase,            // 0:alu, 1:tex,

   // local registers
   // - per chip
   inst_base_vtx,        // vertex base
   inst_base_pix,        // pixel base

   // - per context
   vs_program_base_set,  // connected to SQ_VS_PROGRAM.BASE (12 bits)
   ps_program_base_set,  // connected to SQ_PS_PROGRAM.BASE (12 bits)

   vs_export_count_set,  // connected to SQ_PROGRAM_CNTL.VS_EXPORT_COUNT (4 bits)
   vs_export_mode_set,   // connected to SQ_PROGRAM_CNTL.VS_EXPORT_MODE (3 bits)
   ps_export_mode_set,   // connected to SQ_PROGRAM_CNTL.PS_EXPORT_MODE (3 bits)

   // thread arbiter input
   arb_rts,              //
   arb_state,            //
   arb_status,           //
   arb_thread_type,      // vertex or pixel
   cfs_rtr_q,            // CFS can take a new packet
…
```

Sq_alu_instr_seq.v

```
////////////////////////////////////////////////////////////////////////////////
// sq_alu_instr_seq.v
//
// - receives instruction from alu instr queue (AIQ)
// - reads constants (but data goes directly to ais_output mux)
// - sends instruction to SP over 4 cycles (starting on the correct phase)
…
input [1:0] aiq_export_info;     // {exp_id, pulse_sx}
   input [0:0] aiq_last_in_group;        // last instruction flag
   input [0:0] aiq_last_in_shader;       // last instruction flag
   input [0:0] aiq_thread_type;  // 0: pixel, 1: vertex (shows we operate on either pixel or vertex)
   input [2:0] aiq_context_id;   // context_id (from ctl packet)
   input [5:0] aiq_thread_id;    // clause number
…
// - recall that a 0 here means src is a constant (while 1 means src is a gpr)…

   wire ca_fetch = ~aiq_instr[95];
   wire cb_fetch = ~aiq_instr[94];
   wire cc_fetch = ~aiq_instr[93];

   // - instruction bits 63:61 are used as the const addr msb (these bits are decoded and replaced
   //   before entering the AIQ

   wire [8:0] ca_addr = {aiq_instr[63], aiq_instr[87:80]};
   wire [8:0] cb_addr = {aiq_instr[62], aiq_instr[79:72]};
   wire [8:0] cc_addr = {aiq_instr[61], aiq_instr[71:64]};
…
// ----------------------------
   // -- Input Staging Register –
   // ----------------------------
   // - need to send the vector type and the thread_id back to the thread buffers when
   //   the all the instructions we wanted to run for this thread are done (this will
   //   cause the thread to become valid again)
   // - register this info in from the AIQ on an AIQ pop in order to hold it until the
   //   AIS is done
…
         case (ais_current_state)
           AIS0:
           // - wait until this machine is started by the AIQ read SM
           // - write OSR data into thread buff on new thread (when there was a previous thread…)
           //   - ais_done does updates the thread_buff and clears the alu_instr_pending status bit
           //   - don't assert ais_done yet if the previous instr was a pred set (wait for the pred set
           //     data to arrive from the SP)
           begin
              ais_instr_stall = HI;

              if ( ais_start )
                begin
                   //if ( aiq_new_thread & osr_valid_q & ~osr_pred_set_flag_q ) ais_done = HI;

                   ais_instr_start = HI;
                   ais_instr_stall = LO;
                   ais_next_state = AIS1;
                end
           end

           AIS1: begin ais_next_state = AIS2; end

           AIS2: begin ais_next_state = AIS3; end

           AIS3: begin ais_next_state = AIS4; end

           // ** the AIQ was just popped by the ACS SM, so now must use info saved in ISR ** //

           AIS4: begin ais_next_state = AIS5; end
```

```
          AIS5: begin ais_next_state = AIS6; end

          AIS6:
          begin
             // - the pred set data is loaded now from the previous instr, so assert done now
             // - also write new predicate data into predicate register (in ais_output )

             if ( isr_new_thread_q & osr_pred_set_flag_q ) ais_done = HI;

             ld_osr = HI;
             ais_next_state = AIS7;
          end

          AIS7:
          // - pop the thread off the reservation station buffer when the last instr of the shader is
executed
          // - send free_done when pulse_sx is set, or this is the last instruction of a pixel shader
(since this
          //   is when the pixel export is done)

          begin
             if ( isr_last_in_group_q & ~isr_last_in_shader_q ) ais_done = HI;

             if ( isr_pulse_sx_q ) ais_free_done = HI;   // pixel last logic put into pulse_sx generation

             if ( isr_last_in_shader_q ) ais_pop = HI;

             ais_next_state = AIS0;
          end

        endcase
     end

     // - end ais state machine
```

sq_thread_arb.v

```
// - vertex request priority encoder

   reg [0:0] vtx_winner_vld;
   reg [3:0] vtx_winner;

   always @(vtx_req_q)
     begin
        casez (vtx_req_q)
          16'b0000_0000_0000_0000: begin vtx_winner_vld = LO; vtx_winner = 4'hf; end
          16'b1000_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hf; end
          16'b?100_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'he; end
          16'b??10_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hd; end
          16'b???1_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hc; end
          16'b????_1000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hb; end
          16'b????_?100_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'ha; end
          16'b????_??10_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h9; end
          16'b????_???1_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h8; end
          16'b????_????_1000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h7; end
          16'b????_????_?100_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h6; end
          16'b????_????_??10_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h5; end
          16'b????_????_???1_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h4; end
          16'b????_????_????_1000: begin vtx_winner_vld = HI; vtx_winner = 4'h3; end
          16'b????_????_????_?100: begin vtx_winner_vld = HI; vtx_winner = 4'h2; end
          16'b????_????_????_??10: begin vtx_winner_vld = HI; vtx_winner = 4'h1; end
          16'b????_????_????_???1: begin vtx_winner_vld = HI; vtx_winner = 4'h0; end
          default:                 begin vtx_winner_vld = X;  vtx_winner = 4'bxxxx; end
        endcase
     end


   // - pixel request priority encoder
```

```
   reg [0:0] pix_winner_vld;
   reg [3:0] pix_winner;

   always @(pix_req_q)
     begin
        casez (pix_req_q)
          //16'b0000_0000_0000_0000: begin pix_winner_vld = LO; pix_winner = 4'hf; end
          16'b1000_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hf; end
          16'b?100_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'he; end
          16'b??10_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hd; end
          16'b???1_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hc; end
          16'b????_1000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hb; end
          16'b????_?100_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'ha; end
          16'b????_??10_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'h9; end
          16'b????_???1_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'h8; end
          16'b????_????_1000_0000: begin pix_winner_vld = HI; pix_winner = 4'h7; end
          16'b????_????_?100_0000: begin pix_winner_vld = HI; pix_winner = 4'h6; end
          16'b????_????_??10_0000: begin pix_winner_vld = HI; pix_winner = 4'h5; end
          16'b????_????_???1_0000: begin pix_winner_vld = HI; pix_winner = 4'h4; end
          16'b????_????_????_1000: begin pix_winner_vld = HI; pix_winner = 4'h3; end
          16'b????_????_????_?100: begin pix_winner_vld = HI; pix_winner = 4'h2; end
          16'b????_????_????_??10: begin pix_winner_vld = HI; pix_winner = 4'h1; end
          16'b????_????_????_???1: begin pix_winner_vld = HI; pix_winner = 4'h0; end
          //default:                 begin pix_winner_vld = X;  pix_winner = 4'bxxxx; end
          default:                 begin pix_winner_vld = LO;  pix_winner = 4'bxxxx; end
        endcase
     end


   // - if cfs1 is enabled, alternate btwn rts0 and rts1
   // - if cfs1 is disabled, mask rts1 and always use rts0
   // - what is the algorithm here?  really want to send the thread to the CFS that's available (default
   //   to cfs0 if both are available)
   //   - so getting rid of forced toggle btwn cfs0 and cfs1 - remember to to comment out cfs_turn

   // - there is only one winner max per cycle, so only one of the two RTSs is active in one cycle
   // - it doesn't matter which ALU pipe is used to process a thread, as long as threads are processed in
order
   //   of being selected by the arbiter (i.e. there should be no way for a thread in one ALU pipe to pass
a thread
   //   in the other ALU pipe when they are from the same context)

   //assign arb_rts0 = arb_rts & (~cfs_turn | ~cfs1_enable);
   //assign arb_rts1 = arb_rts &  cfs_turn & cfs1_enable;

   //wire [0:0] cfs_rtr = cfs_rtr0 | cfs_rtr1;

   wire [0:0] send_to_cfs0 = cfs_rtr0;
   wire [0:0] send_to_cfs1 = ~cfs_rtr0 & cfs_rtr1 & cfs1_enable;

   assign     arb_rts0 = arb_rts & send_to_cfs0;
   assign     arb_rts1 = arb_rts & send_to_cfs1;

   wire [0:0] arb_xfc0 = arb_rts0 & cfs_rtr0;
   wire [0:0] arb_xfc1 = arb_rts1 & cfs_rtr1;

   wire [0:0] arb_xfc = arb_xfc0 | arb_xfc1;


   // --------------------
   // -- Arb Output Mux --
   // --------------------
   // - choose between tex state/status and pix state/status depending on overall winner
   // - vtx tex has no lod
   // - vtx alu has no lod
   // - pix tex does have LOD (PIX_CTL_PKT_WIDTH and CTL_PKT_WIDTH have lod)
   // - pix alu has no lod
```

```
   always @(type_winner_q or vtx_state or pix_state)
    begin
      //arb_state = {STATE_WIDTH{LO}};
      case (type_winner_q)
        HI: arb_state = vtx_state;    // these are unequal - msb's get 0's by above assignment
        LO: arb_state = pix_state;
        //default: arb_state = {STATE_WIDTH{X}};
      endcase
    end

   always @(type_winner_q or vtx_status or pix_status)
    begin
      //arb_status = {STATUS_WIDTH{LO}};
      case (type_winner_q)
        HI: arb_status = vtx_status;
        LO: arb_status = pix_status;
        //default: arb_status = {STATUS_WIDTH{X}};
      endcase
    end
```

sq_shader_seq.v

```
// shader_seq.v
//
// - instantiates 16 reservation stations
//
// issues:
// -
//
////////////////////////////////////////////////////////////////////////////////

module sq_shader_seq
(
  shader_seq_type,      // a strap that tells this module if it's a vertex or pixel shader seq

  // control packet input
  input_cp,             // control packet data from the input SM
  input_rts,            // rts from the input SM
  input_rtr,            // rtr from texture RS0

  // texture clause arbiter interface
  tex_req,              // 8 texture RS requests
  tex_cp,         // vector of 8 control packets
  tex_rtr,              // 8 RTSs (not fulls) to the ALU arbiters
  tca_winner_ack,       // 8 ack bits from arb - only the winner bit is set
  tca_empty_ack,        // 8 ack bits from arb - each empty requesting clause is ack'd to move it to next
RS

  TP_SQ_data_rdy,       // data ready indicator from TPC - increment the alu RS counter
  TP_SQ_type,           // the vector type: pixel=0, vertex=1
  TP_SQ_clause_num,  // the alu RS number whose count should be incremented

  // alu clause arbiter interface
  alu_req,              //
  alu_cp,               //
  alu_rtr,              //
  aca_winner_ack,       //
  aca_empty_ack,        //
```

```
  ais0_data_rdy,          // done indicator from AIS0 - increment the tex RS counter
  ais0_vector_type,               // the vector type: pixel=0, vertex=1
  ais0_clause_num,      // the tex RS number whose count should be incremented

  ais1_data_rdy,          // done indicator from AIS1 - increment the tex RS counter
  ais1_vector_type,               // the vector type: pixel=0, vertex=1
  ais1_clause_num,      // the tex RS number whose count should be incremented

  // exit SM interface
  state_change,// a pulse high indicates that the state exiting the SS has changed
  old_state,              // the state that has finished (because a new state has emerged)
  dealloc_req,            // request to deallocate GPRs
  dealloc_ack,            // the dealloc request has been acknowleged

  clk,
  reset
);


  // -- parameters --

  parameter CP_WIDTH = 8;
  parameter STATE_WIDTH = 3;

  parameter FIFO_WIDTH = CP_WIDTH;
  parameter FIFO_DEPTH = 4;
  parameter FIFO_ADDR_BITS = 2;

  parameter LO = 1'b0;
  parameter HI = 1'b1;
  parameter X  = 1'bx;


  // ------------------------------------------------------------------------------
  // -- ios --
  // ------------------------------------------------------------------------------

  input    shader_seq_type;

  input  [CP_WIDTH-1:0]         input_cp;
  input                         input_rts;
  output                        input_rtr;

  output [8:0]                  tex_req;
  output [8*CP_WIDTH-1:0]       tex_cp;
  output [8:1]                  tex_rtr;

  input  [7:0]                  tca_winner_ack;
  input  [7:0]                  tca_empty_ack;

  input [0:0]       TP_SQ_data_rdy;
  input [0:0]   TP_SQ_type;
  input [2:0]       TP_SQ_clause_num;

  output [7:0]                  alu_req;
```

```
output [8*CP_WIDTH-1:0]      alu_cp;
output [7:0]                 alu_rtr;

input  [7:0]                 aca_winner_ack;
input  [7:0]                 aca_empty_ack;

input                ais0_data_rdy;
input                ais0_vector_type;
input  [2:0]                 ais0_clause_num;

input                ais1_data_rdy;
input                ais1_vector_type;
input  [2:0]                 ais1_clause_num;

output                       state_change;
output [2:0]                 old_state;
output                       dealloc_req;
input                dealloc_ack;

input                clk;
input                reset;

// - output register declarations

//reg [8:0]                  tex_req;
//reg [7:0]                  alu_req;


// ---------------------------------------------------------------------------
// -- internal signals --
// ---------------------------------------------------------------------------

wire [CP_WIDTH-1:0]          tex_ctl_pkt0;
wire [CP_WIDTH-1:0]          tex_ctl_pkt1;
wire [CP_WIDTH-1:0]          tex_ctl_pkt2;
wire [CP_WIDTH-1:0]          tex_ctl_pkt3;
wire [CP_WIDTH-1:0]          tex_ctl_pkt4;
wire [CP_WIDTH-1:0]          tex_ctl_pkt5;
wire [CP_WIDTH-1:0]          tex_ctl_pkt6;
wire [CP_WIDTH-1:0]          tex_ctl_pkt7;
wire [CP_WIDTH-1:0]          alu_ctl_pkt0;
wire [CP_WIDTH-1:0]          alu_ctl_pkt1;
wire [CP_WIDTH-1:0]          alu_ctl_pkt2;
wire [CP_WIDTH-1:0]          alu_ctl_pkt3;
wire [CP_WIDTH-1:0]          alu_ctl_pkt4;
wire [CP_WIDTH-1:0]          alu_ctl_pkt5;
wire [CP_WIDTH-1:0]          alu_ctl_pkt6;
wire [CP_WIDTH-1:0]          alu_ctl_pkt7;


// group all the control packets together into one big vector for output to the arbiter

wire [8*CP_WIDTH-1:0] tex_cp = {tex_ctl_pkt7, tex_ctl_pkt6, tex_ctl_pkt5, tex_ctl_pkt4,
                                tex_ctl_pkt3, tex_ctl_pkt2, tex_ctl_pkt1, tex_ctl_pkt0};

wire [8*CP_WIDTH-1:0] alu_cp = {alu_ctl_pkt7, alu_ctl_pkt6, alu_ctl_pkt5, alu_ctl_pkt4,
```

```
                                alu_ctl_pkt3, alu_ctl_pkt2, alu_ctl_pkt1, alu_ctl_pkt0};

reg [0:0] tpc_data_rdy;
reg [0:0] tpc_type;
reg [2:0] tpc_clause_num;


// ------------------------------------------------------------------------------
// -- combinational logic --
// ------------------------------------------------------------------------------

// - select the RS counter to increment based on clause number sent by TPC/AIS
// - counts represent the number of valid entries in a RS FIFO; because ctl packets are
//   moved into the next RS before the vector they represent has actually finished, the
//   count is used to gate the requests to the next arbiter until the clause is actually
//   done
// - this is a decoder enabled by data_rdy

reg [7:0] tpc_cnt_inc;
reg [7:0] ais0_cnt_inc;
reg [7:0] ais1_cnt_inc;

always @(tpc_data_rdy or tpc_clause_num or tpc_type or shader_seq_type)
  begin
        tpc_cnt_inc = 8'h00;
        if (tpc_data_rdy & (tpc_type == shader_seq_type))
          tpc_cnt_inc[tpc_clause_num] = 1'b1;
  end

always @(ais0_data_rdy or ais0_clause_num or ais0_vector_type or shader_seq_type)
  begin
        ais0_cnt_inc = 8'h00;
        if (ais0_data_rdy & (ais0_vector_type == shader_seq_type))
          ais0_cnt_inc[ais0_clause_num] = 1'b1;
  end

always @(ais1_data_rdy or ais1_clause_num or ais1_vector_type or shader_seq_type)
  begin
        ais1_cnt_inc = 8'h00;
        if (ais1_data_rdy & (ais1_vector_type == shader_seq_type))
          ais1_cnt_inc[ais1_clause_num] = 1'b1;
  end

wire [7:0] ais_cnt_inc = ais0_cnt_inc | ais1_cnt_inc;


// - create the RS request by masking the RS FIFO rts when the associated RS count is zero
// - this is done because a control packet is moved to the next RS before the RS can actually tell
//   the arbiter about it
//   - in both cases, in order to facilitate the advancement of empty clauses, the packet is moved
//     to the next RS when the arbiter selects it
//   - in the case of alu RSs, the TPC must indicate that the texture data has been loaded into the
//     GPRs before incrementing the count
//   - in the case of tex RSs, the AIS will increment the count when it's done

//wire [FIFO_ADDR_BITS-1:0]           tex_count [0:8];  // tex_count[8] is for the exit RS
```

```
//wire [FIFO_ADDR_BITS-1:0]           alu_count [0:7];
wire [FIFO_ADDR_BITS-1:0]  tex_count0;
wire [FIFO_ADDR_BITS-1:0]  tex_count1;
wire [FIFO_ADDR_BITS-1:0]  tex_count2;
wire [FIFO_ADDR_BITS-1:0]  tex_count3;
wire [FIFO_ADDR_BITS-1:0]  tex_count4;
wire [FIFO_ADDR_BITS-1:0]  tex_count5;
wire [FIFO_ADDR_BITS-1:0]  tex_count6;
wire [FIFO_ADDR_BITS-1:0]  tex_count7;
wire [FIFO_ADDR_BITS-1:0]  tex_count8;
wire [FIFO_ADDR_BITS-1:0]  alu_count0;
wire [FIFO_ADDR_BITS-1:0]  alu_count1;
wire [FIFO_ADDR_BITS-1:0]  alu_count2;
wire [FIFO_ADDR_BITS-1:0]  alu_count3;
wire [FIFO_ADDR_BITS-1:0]  alu_count4;
wire [FIFO_ADDR_BITS-1:0]  alu_count5;
wire [FIFO_ADDR_BITS-1:0]  alu_count6;
wire [FIFO_ADDR_BITS-1:0]  alu_count7;

wire [8:0]                 tex_rts;        // tex_rts[8] is for the exit RS
wire [7:0]                 alu_rts;

// - this could be done in the reservation station...
//always @(tex_rts or tex_count)
//  for (i=0; i<9; i=i+1) begin
//     tex_req[i] = tex_rts[i] & |(tex_count[i]);
//  end

//always @(alu_rts or alu_count)
//  for (i=0; i<8; i=i+1) begin
//     alu_req[i] = alu_rts[i] & |(alu_count[i]);
//  end

assign tex_req[0] = tex_rts[0] & |tex_count0;
assign tex_req[1] = tex_rts[1] & |tex_count1;
assign tex_req[2] = tex_rts[2] & |tex_count2;
assign tex_req[3] = tex_rts[3] & |tex_count3;
assign tex_req[4] = tex_rts[4] & |tex_count4;
assign tex_req[5] = tex_rts[5] & |tex_count5;
assign tex_req[6] = tex_rts[6] & |tex_count6;
assign tex_req[7] = tex_rts[7] & |tex_count7;
assign tex_req[8] = tex_rts[8] & |tex_count8;
assign alu_req[0] = alu_rts[0] & |alu_count0;
assign alu_req[1] = alu_rts[1] & |alu_count1;
assign alu_req[2] = alu_rts[2] & |alu_count2;
assign alu_req[3] = alu_rts[3] & |alu_count3;
assign alu_req[4] = alu_rts[4] & |alu_count4;
assign alu_req[5] = alu_rts[5] & |alu_count5;
assign alu_req[6] = alu_rts[6] & |alu_count6;
assign alu_req[7] = alu_rts[7] & |alu_count7;

// - the acknowledge to a RS is the OR of the winner and empty ack vectors
//   - the ack advances the ctl packet to the next RS
//   - want to advance when either the clause was picked by the arbiter or when
//     the clause is empty (no instructions)
```

```
wire [7:0] tca_ack = tca_winner_ack | tca_empty_ack;

//wire [7:0] aca_winner_ack = aca0_winner_ack | aca1_winner_ack;
//wire [7:0] aca_empty_ack  = aca0_empty_ack  | aca1_empty_ack;
wire [7:0] aca_ack        = aca_winner_ack  | aca_empty_ack;


// ----------------------------------------------------------------------------
// -- registers --
// ----------------------------------------------------------------------------

// - block input registers for signals from TPC

always @(posedge clk)
  begin
        tpc_data_rdy   <= TP_SQ_data_rdy;
        tpc_type       <= TP_SQ_type;
        tpc_clause_num <= TP_SQ_clause_num;
  end


// ----------------------------------------------------------------------------
// -- state machines --
// ----------------------------------------------------------------------------


// ----------------------------------------------------------------------------
// -- module instatiations --
// ----------------------------------------------------------------------------

// 16 reservation stations: 8 texture, 8 alu
// - the RSs are connected tex to alu to tex etc., with an exit RS connected after alu rs7 (like tex rs8)
// - the write rts/rtr for tex rs0 is from the input sm
// - the read rts's are qualified with the RS count and sent to the arbiter
// - the arbiter sends an ack which rtr's the sender and rts's the receiver (i.e. next RS)
// - the next RS rtr goes back to the arbiter and must be high to enable a grant

wire tex_rs0_cnt_inc = input_rts & input_rtr;

res_station // tex rs0
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs0
  (.write_rts(input_rts),  .write_rtr(input_rtr),       .write_data(input_cp),
   .read_rts (tex_rts[0]), .read_rtr (tca_ack[0]),      .read_data (tex_ctl_pkt0),
   .empty_inc(LO),         .count_inc(tex_rs0_cnt_inc), .count(tex_count0),
   .clk(clk), .reset(reset)
  );
res_station // alu rs0
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs0
  (.write_rts(tca_ack[0]),       .write_rtr(alu_rtr[0]),     .write_data(tex_ctl_pkt0),
   .read_rts (alu_rts[0]),       .read_rtr (aca_ack[0]),     .read_data (alu_ctl_pkt0),
   .empty_inc(tca_empty_ack[0]), .count_inc(tpc_cnt_inc[0]), .count(alu_count0),
   .clk(clk), .reset(reset)
  );
res_station // tex rs1
```

```
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs1
  (.write_rts(aca_ack[0]),       .write_rtr(tex_rtr[1]),     .write_data(alu_ctl_pkt0),
   .read_rts (tex_rts[1]),       .read_rtr (tca_ack[1]),     .read_data (tex_ctl_pkt1),
   .empty_inc(aca_empty_ack[0]), .count_inc(ais_cnt_inc[0]), .count(tex_count1),
   .clk(clk), .reset(reset)
  );
res_station // alu rs1
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs1
  (.write_rts(tca_ack[1]),       .write_rtr(alu_rtr[1]),       .write_data(tex_ctl_pkt1),
   .read_rts (alu_rts[1]),       .read_rtr(aca_ack[1]),  .read_data (alu_ctl_pkt1),
   .empty_inc(tca_empty_ack[1]), .count_inc(tpc_cnt_inc[1]), .count(alu_count1),
   .clk(clk), .reset(reset)
  );
res_station // tex rs2
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs2
  (.write_rts(aca_ack[1]), .write_rtr(tex_rtr[2]), .write_data(alu_ctl_pkt1),
   .read_rts (tex_rts[2]), .read_rtr(tca_ack[2]),  .read_data (tex_ctl_pkt2),
   .empty_inc(aca_empty_ack[1]), .count_inc(ais_cnt_inc[1]), .count(tex_count2),
   .clk(clk), .reset(reset)
  );
res_station // alu rs2
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs2
  (.write_rts(tca_ack[2]), .write_rtr(alu_rtr[2]), .write_data(tex_ctl_pkt2),
   .read_rts (alu_rts[2]), .read_rtr(aca_ack[2]),  .read_data (alu_ctl_pkt2),
   .empty_inc(tca_empty_ack[2]), .count_inc(tpc_cnt_inc[2]), .count(alu_count2),
   .clk(clk), .reset(reset)
  );
res_station // tex rs3
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs3
  (.write_rts(aca_ack[2]), .write_rtr(tex_rtr[3]), .write_data(alu_ctl_pkt2),
   .read_rts (tex_rts[3]), .read_rtr(tca_ack[3]),  .read_data (tex_ctl_pkt3),
   .empty_inc(aca_empty_ack[2]), .count_inc(ais_cnt_inc[2]), .count(tex_count3),
   .clk(clk), .reset(reset)
  );
res_station // alu rs3
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs3
  (.write_rts(tca_ack[3]), .write_rtr(alu_rtr[3]), .write_data(tex_ctl_pkt3),
   .read_rts (alu_rts[3]), .read_rtr(aca_ack[3]),  .read_data (alu_ctl_pkt3),
   .empty_inc(tca_empty_ack[3]), .count_inc(tpc_cnt_inc[3]), .count(alu_count3),
   .clk(clk), .reset(reset)
  );
res_station // tex rs4
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs4
  (.write_rts(aca_ack[3]), .write_rtr(tex_rtr[4]), .write_data(alu_ctl_pkt3),
   .read_rts (tex_rts[4]), .read_rtr(tca_ack[4]),  .read_data (tex_ctl_pkt4),
   .empty_inc(aca_empty_ack[3]), .count_inc(ais_cnt_inc[3]), .count(tex_count4),
   .clk(clk), .reset(reset)
  );
res_station // alu rs4
```

```
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs4
  (.write_rts(tca_ack[4]), .write_rtr(alu_rtr[4]), .write_data(tex_ctl_pkt4),
   .read_rts (alu_rts[4]), .read_rtr(aca_ack[4]),  .read_data (alu_ctl_pkt4),
   .empty_inc(tca_empty_ack[4]), .count_inc(tpc_cnt_inc[4]), .count(alu_count4),
   .clk(clk), .reset(reset)
  );
res_station // tex rs5
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs5
  (.write_rts(aca_ack[4]), .write_rtr(tex_rtr[5]), .write_data(alu_ctl_pkt4),
   .read_rts (tex_rts[5]), .read_rtr(tca_ack[5]),  .read_data (tex_ctl_pkt5),
   .empty_inc(aca_empty_ack[5]), .count_inc(ais_cnt_inc[4]), .count(tex_count5),
   .clk(clk), .reset(reset)
  );
res_station // alu rs5
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs5
  (.write_rts(tca_ack[5]), .write_rtr(alu_rtr[5]), .write_data(tex_ctl_pkt5),
   .read_rts (alu_rts[5]), .read_rtr(aca_ack[5]),  .read_data (alu_ctl_pkt5),
   .empty_inc(tca_empty_ack[4]), .count_inc(tpc_cnt_inc[5]), .count(alu_count5),
   .clk(clk), .reset(reset)
  );
res_station // tex rs6
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs6
  (.write_rts(aca_ack[5]), .write_rtr(tex_rtr[6]), .write_data(alu_ctl_pkt5),
   .read_rts (tex_rts[6]), .read_rtr(tca_ack[6]),  .read_data (tex_ctl_pkt6),
   .empty_inc(aca_empty_ack[5]), .count_inc(ais_cnt_inc[5]), .count(tex_count6),
   .clk(clk), .reset(reset)
  );
res_station // alu rs6
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs6
  (.write_rts(tca_ack[6]), .write_rtr(alu_rtr[6]), .write_data(tex_ctl_pkt6),
   .read_rts (alu_rts[6]), .read_rtr(aca_ack[6]),  .read_data (alu_ctl_pkt6),
   .empty_inc(tca_empty_ack[6]), .count_inc(tpc_cnt_inc[6]), .count(alu_count6),
   .clk(clk), .reset(reset)
  );
res_station // tex rs7
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_tex_rs7
  (.write_rts(aca_ack[6]), .write_rtr(tex_rtr[7]), .write_data(alu_ctl_pkt6),
   .read_rts (tex_rts[7]), .read_rtr(tca_ack[7]),  .read_data (tex_ctl_pkt7),
   .empty_inc(aca_empty_ack[6]), .count_inc(ais_cnt_inc[6]), .count(tex_count7),
   .clk(clk), .reset(reset)
  );
res_station // alu rs7
 #(.DATA_BITS(FIFO_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
u_alu_rs7
  (.write_rts(tca_ack[7]), .write_rtr(alu_rtr[7]), .write_data(tex_ctl_pkt7),
   .read_rts (alu_rts[7]), .read_rtr(aca_ack[7]),  .read_data (alu_ctl_pkt7),
   .empty_inc(tca_empty_ack[7]), .count_inc(tpc_cnt_inc[7]), .count(alu_count7),
   .clk(clk), .reset(reset)
  );
```

```
  wire [2:0] new_state;

  // exit RS
  res_station
   #(.DATA_BITS(STATE_WIDTH), .NUM_WORDS(FIFO_DEPTH), .ADDR_BITS(FIFO_ADDR_BITS))
  u_tex_rs8
    (.write_rts(aca_ack[7]), .write_rtr(tex_rtr[8]), .write_data(alu_ctl_pkt7[STATE_WIDTH-1:0]),
     .read_rts (tex_rts[8]), .read_rtr(exit_sm_rtr), .read_data (new_state),
     .empty_inc(aca_empty_ack[7]), .count_inc(ais_cnt_inc[7]), .count(tex_count8),
     .clk(clk), .reset(reset)
    );

  // ------------------------
  // -- exit state machine --
  // ------------------------

  exit_sm
  u_exit_sm
  (
    .new_state_rts(tex_req[8]),
    .new_state_rtr(exit_sm_rtr),
    .new_state(new_state),

    .state_diff(state_change),
    .old_state_q(old_state),

    .dealloc_req(dealloc_req),
    .dealloc_ack(dealloc_ack),

    .clk(clk),
    .reset(reset)
  );


endmodule
```

Electronic Patent Application Fee Transmittal

Application Number:	13109738
Filing Date:	17-May-2011
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Filer:	Christopher J. Reckamp/Christine Wright
Attorney Docket Number:	00100.36.0001

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Extension - 3 months with $0 paid	1253	1	1270	1270

TCL 1002

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
			Total in USD ($)	**1270**

Electronic Acknowledgement Receipt

EFS ID:	11860180
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp/Christine Wright
Filer Authorized By:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001
Receipt Date:	18-JAN-2012
Filing Date:	17-MAY-2011
Time Stamp:	12:01:42
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$1270
RAM confirmation Number	11637
Deposit Account	020390
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		360001_Response.pdf	120979 5ea628e325d222b0f8f3ef2eed1ab6f5fd856460	yes	8

Multipart Description/PDF files in .zip description

Document Description	Start	End
Amendment/Req. Reconsideration-After Non-Final Reject	1	1
Claims	2	5
Applicant Arguments/Remarks Made in an Amendment	6	8

Warnings:					
Information:					
2	Miscellaneous Incoming Letter	360001_Declaration.pdf	124271 bbb6bc0b6e0428a869fcb8a54972857d38d43dd7	no	6
Warnings:					
Information:					
3	Miscellaneous Incoming Letter	360001_ExhibitA.pdf	738779 1b5bd3d447e9b644e4b83f48ef84eaeb5f3fc50d	no	93
Warnings:					
Information:					
4	Miscellaneous Incoming Letter	360001_ExhibitB.pdf	260167 23c581deb4a1ade2018254c5a4bf3ca4d0c519a1	no	20
Warnings:					
Information:					
5	Fee Worksheet (SB06)	fee-info.pdf	30614 156554244122c7c58b7a8c2f31ad5fe86e79a08a	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	1274810		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

29153 7590 03/15/2012

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

EXAMINER	
WASHBURN, DANIEL C	

ART UNIT	PAPER NUMBER
2628	

NOTIFICATION DATE	DELIVERY MODE
03/15/2012	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

inteas@faegrebd.com
cynthia.payson@faegredb-.com

Office Action Summary	Application No.	Applicant(s)	
	13/109,738	MOREIN ET AL.	
	Examiner	Art Unit	
	DANIEL WASHBURN	2628	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *18 January 2012*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

5)☒ Claim(s) *1-16* is/are pending in the application.

5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6)☐ Claim(s) _____ is/are allowed.

7)☒ Claim(s) *1-16* is/are rejected.

8)☐ Claim(s) _____ is/are objected to.

9)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

10)☒ The specification is objected to by the Examiner.

11)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

12)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

13)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Specification

Applicant is reminded of the proper language and format for an abstract of the disclosure.

The abstract should be in narrative form and generally limited to a single paragraph on a separate sheet within the range of 50 to 150 words. It is important that the abstract not exceed 150 words in length since the space provided for the abstract on the computer tape used by the printer is limited. **The form and legal phraseology often used in patent claims, such as "means" and "said," should be avoided.** The abstract should describe the disclosure sufficiently to assist readers in deciding whether there is a need for consulting the full patent text for details.

The language should be clear and concise and should not repeat information given in the title. It should avoid using phrases which can be implied, such as, "The disclosure concerns," "The disclosure defined by this invention," "The disclosure describes," etc.

Declaration filed under 37 CFR 1.131

The declaration filed 1/18/12 under 37 CFR 1.131 has been considered but is ineffective to overcome the prior art reference Lindholm (US 7,038,685, "the Lindholm reference").

The declaration does not meet the requirements of 37 CFR 1.131 section (a).

37 CFR 1.131 section (a) states (in relevant part):

"(a) When any claim of an application or a patent under reexamination is rejected, the inventor of the subject matter of the rejected claim, the owner of the patent under reexamination, or the party qualified under §§ 1.42, 1.43, or 1.47, may submit an appropriate oath or declaration to establish invention of the subject matter of the rejected claim prior to the effective date of the reference or activity on which the rejection is based. The effective date of a U.S. patent, U.S. patent application publication, or international application publication under PCT Article 21(2) is the earlier of its publication date or date that it is effective as a reference under 35 U.S.C. 102(e). **Prior invention may not be established under this section in any country other than the United States, a NAFTA country, or a WTO member country. Prior invention may not be established under this section before December 8, 1993, in**

a NAFTA country other than the United States, or before January 1, 1996, in a WTO member country other than a NAFTA country." (emphasis added)

Section 2 of Applicants' declaration describes (in relevant part):

"2. We conceived the Invention prior to June 30, 2003 while employed by ATI Technologies Inc. and/or one of its wholly owned subsidiaries ("ATI") as indicated by attached Exhibits A and B ... Prior to June 30, 2003 we created a graphics processing system that operated as claimed using a computer system that successfully executed the Model Code. Prior to June 30, 2003 we also created a graphics processing system as claimed in the form of a computer system that used an RTL simulator to successfully validate the operation of an integrated circuit version of the claimed graphics processing system and method."

As quoted from Applicants' declaration, section 2 describes conception and reduction to practice of the claimed invention prior to June 30, 2003. Section 2 further describes that the conception and reduction to practice of the claimed invention was carried out while the inventors were employed by ATI Technologies Inc. and/or one of its wholly owned subsidiaries.

However, section 2, and the declaration as a whole, fails to specify whether or not the conception and reduction to practice was carried out in the United States, a NAFTA country, or a WTO member country. As quoted from 37 CFR 1.131 section (a), "[p]rior invention may not be established under this section in any country other than the United States, a NAFTA country, or a WTO memory country". Thus, the declaration is ineffective to overcome the Lindholm reference due to this first deficiency.

Further, the declaration does not meet the requirements of 37 CFR 1.131 section (b).

37 CFR 1.131 section (b) states:

"(b) The showing of facts shall be such, in character and weight, as to establish reduction to practice prior to the effective date of the reference, or conception

of the invention prior to the effective date of the reference coupled with due diligence from prior to said date to a subsequent reduction to practice or to the filing of the application. Original exhibits of drawings or records, or photocopies thereof, must accompany and form part of the affidavit or declaration or their absence must be satisfactorily explained."

MPEP 715.07 [R-3] "Facts and Documentary Evidence", section I. "General Requirements", offers further guidance regarding the requirements of 37 CFR 1.131 section (b).

MPEP 715.07, section I., describes (in relevant part):

"The essential thing to be shown under 37 CFR 1.131 is priority of invention and this may be done by any satisfactory evidence of the fact. FACTS, not conclusions, must be alleged. Evidence in the form of exhibits may accompany the affidavit or declaration. Each exhibit relied upon should be specifically referred to in the affidavit or declaration, in terms of what it is relied upon to show … when reviewing a 37 CFR 1.131 affidavit or declaration, the examiner must consider all of the evidence presented in its entirety, including the affidavits or declarations and all accompanying exhibits, records and "notes." An accompanying exhibit need not support all claimed limitations, provided that any missing limitation is supported by the declaration itself. Ex parte Ovshinsky, 10 USPQ2d 1075 (Bd. Pat. App. & Inter. 1989).

The affidavit or declaration and exhibits must clearly explain which facts or data applicant is relying on to show completion of his or her invention prior to the particular date. Vague and general statements in broad terms about what the exhibits describe along with a general assertion that the exhibits describe a reduction to practice "amounts essentially to mere pleading, unsupported by proof or a showing of facts" and, thus, does not satisfy the requirements of 37 CFR 1.131(b). In re Borkowski, 505 F.2d 713, 184 USPQ 29 (CCPA 1974). **Applicant must give a clear explanation of the exhibits pointing out exactly what facts are established and relied on by applicant.** 505 F.2d at 718-19, 184 USPQ at 33. See also In re Harry, 333 F.2d 920, 142 USPQ 164 (CCPA 1964) (Affidavit "asserts that facts exist but does not tell what they are or when they occurred.")." (emphasis added)

Section 2 of Applicants' declaration describes (in relevant part):

"Prior to June 30, 2003 we created a graphics processing system that operated as claimed using a computer system that successfully executed the Model Code. Prior to June 30, 2003 we also created a graphics processing system as claimed in the form of a computer system that used an RTL simulator to successfully validate the operation

of an integrated circuit version of the claimed graphics processing system and method At least the following language and citations adequately support the above:

a. As shown in Exhibit A, the Model Code comprises various software instructions written in the well-known C++ language. When executed by the computer system, the Model Code caused the computer system to operate as claimed in at least claims 1-5, 12 and 15 of the Invention.

b. Using the Model Code, we successfully verified the operation of the claimed subject matter for its intended purpose through emulation thereof.

c. As shown in Exhibit B, the Chip Design Code comprises various instructions written in a well-known hardware description language. The Chip Design Code was used by an RTL simulator system to validate the operation of an integrated circuit version of the claimed graphics processing system and method as claimed in at least claims 1-5, 12 and 15. As further known by practitioners in the field of integrated circuit design, such instructions are used to generate gate level detail for silicon fabrication.

d. On information and belief, the computer system operating the Model Code and the RTL simulator system operating the Chip Design Code represents the claimed structure and operation embodied in an integrated graphics processing circuit chip referred to as the ATI XENOS chip produced by ATI on or about October, 2004 that was incorporated in the XBOX 360 product.

Accordingly, the contents of Exhibits A and B establish the possession by us of the whole Invention, failing within the scope of currently pending claims, such as but not limited to at least claims 1-5, 12 and 15."

As quoted from Applicants' declaration, section 2 describes that Exhibit A is Model Code that, when executed by the computer system, caused the computer system to operate as claimed in at least claims 1-5, 12, and 15 of the Invention. Further, section 2 describes that Exhibit B is Chip Design Code that was used by an RTL simulator system to validate operation of an integrated circuit version of the claimed graphics processing system and method as claimed in at least claims 1-5, 12, and 15.

However, section 2, and the declaration as a whole, fails to clearly explain which facts or data applicant is relying on to show completion of his or her invention prior to

the June 30, 2003. The portions of Applicants' declaration quoted above are considered nothing more than vague and general statements in broad terms about what the exhibits describe along with general assertions that the exhibits describe a reduction to practice, which does not satisfy the requirements of 37 CFR 1.131 section (b). Thus, the declaration is ineffective to overcome the Lindholm reference due to this second deficiency.

Regarding claim 1, the Examiner is unable to determine which portions of Exhibit A and/or Exhibit B describe the claimed method steps of “performing vertex manipulation operations and pixel manipulation operations…and continuing pixel calculation operations that are to be or are currently being performed by the processor..."

Regarding claim 2, the Examiner is unable to determine which portions of Exhibit A and/or Exhibit B describe the claimed “unified shader, comprising: a general purpose register block…a processor unit; and a sequencer, coupled to the general purpose register block and the processor unit…”

Regarding claims 3 and 4, the Examiner is unable to determine which portions of Exhibit A and/or Exhibit B describe the claimed “unified shader comprising: a processor unit…and shared resources…the processor unit operative to use the shared resources…”

Regarding claim 5, the Examiner is unable to determine which portions of Exhibit A and/or Exhibit B describe the claimed “unified shader comprising: a processor unit; a sequencer coupled to the processor unit…”

Regarding claim 12, the Examiner is unable to determine which portions of Exhibit A and/or Exhibit B describe the claimed "graphics processor comprising: a unified shader comprising a processor unit…"

Regarding claim 15, the Examiner is unable to determine which portions of Exhibit A and/or Exhibit B describe the claimed "unified shader comprising: a processor unit flexibly controlled…"

Thus, for at least the reasons given above, the declaration filed 1/18/12 under 37 CFR 1.131 is ineffective to overcome the Lindholm reference.

As an additional note, the Examiner would like to point out that US Pat 7,015,913, to Lindholm et al., filed June 27th, 2003, appears, after brief review, to include a disclosure that is similar to US Pat 7,038,685 to Lindholm, which is used in the rejections that follow (see FIG. 2 of each patent). The Examiner has not given Lindholm et al. (US 7,015,913) a thorough review as to whether or not it teaches one or more of Applicants' claims, but it may be worth Applicants' time to review Lindholm et al. (US 7,015,913) and adjust the declaration such that conception and reduction to practice of the claimed invention is declared to have occurred prior to June 27th, 2003 (if such a statement is true), in order to avoid a future rejection based on the teachings of prior art reference Lindholm et al. (US 7,015,913).

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-16 are rejected under 35 U.S.C. 102(e) as being anticipated by Lindholm (US 7,038,685).

RE claim 1, Lindholm describes a method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data (

3:59-65: “Programmable Graphics Processing Pipeline 150 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample or any other data. For simplicity, the remainder of this description will use the term 'samples' to refer to graphics data such as surfaces, primitives, vertices, pixels, fragments, or the like."

6:38-59: “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

7:6-10: “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities".

7:36-43: “Once a thread is assigned to a source sample, the thread is allocated storage resources such as locations in a Register File 350 to retain intermediate data generated during execution of program instructions associated with the thread."

9:33-56: "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

15:7-13: “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, Lindholm describes performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block (sample data, such as vertex or pixel data, is transmitted to Register File 350) and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data (the multi-threaded processing unit 400 carries out vertex operations on vertex data unless the Register File 350 doesn’t have enough room to store the incoming vertex data, in which case the thread associated with the vertex data and vertex operations must wait until enough space becomes available); and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available (

7:6-21: “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

8:15-58: "Thread Selection Unit 415 reads one or more thread entries based on thread execution priorities and outputs selected thread entries to Instruction Cache 410. Instruction cache 410 determines if the program instructions corresponding to the program counters and sample type included in the thread state data for each thread entry are available in Instruction Cache 410 ... The program instructions corresponding to the program counters from the one or more thread entries are output by Instruction

Cache 410 to … Instruction Scheduler 430 … Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU [instruction window unit] 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350. An instruction specifies the location of source data needed to execute the instruction."

15:7-13: “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, Lindholm is considered to describe an embodiment including continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available, as the Execution Unit 470 may be carrying out calculations for one or more high priority pixel threads based on instructions stored in Instruction Cache 410 and/or IWU 435 while a low priority vertex thread is waiting for the one or more pixel threads to finish such that when the pixel threads finish the system will deallocate the resources assigned to the completed pixel threads in the Register File 350 and will allocate the requested amount of resources to the queued up vertex thread).

RE claim 2, Lindholm describes a unified shader, comprising:

a general purpose register block for maintaining data (

7:37-43: “Once a thread is assigned to a source sample, the thread is allocated storage resources such as locations in a Register File 350 to retain intermediate data generated during execution of program instructions associated with the thread.”);

a processor unit (FIG. 4 “Execution Unit 470” and “PCU 375”);

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block (

8:33-9:32 “Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350."

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."); and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs (

9:39-46 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … and output the processed sample to a destination specified by the instruction. The destination may be Vertex Output Buffer 260, Pixel Output Buffer 270, or Register File 350.”

4:42-5:35 “Execution Pipelines 240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates … Execution Pipelines 240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 260 ... Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted ... programmable computation units (PCUs) within an Execution Pipeline 240 ... perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like. Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270."

Thus, the Execution Unit 470 is considered a processor unit that executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs (also see 4:22-5:35)).

RE claim 3, Lindholm describes a unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations (FIG. 4 “Execution Unit 470” and “PCU 375”.

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

Thus, the Execution Unit 470 and internal PCU 375 are collectively considered a processor unit operative to perform vertex calculation operations and pixel calculation operations); and

shared resources, operatively coupled to the processor unit (FIG. 4 illustrates Register File 350 coupled to Execution Unit 470, and 7:37-43 describes that the Register File 350 is shared among threads);

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform

vertex calculation operations (7:37-43, all types of processing threads can use the Register File 350, where thread types include vertex and pixel threads (see 6:43-44).

7:6-36 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, when pixel threads have priority over vertex threads the processor unit will allocate the pixel data to the Register File 350 and will perform pixel calculation operations until enough shared resources become available in the Register File 350 to begin carrying out vertex threads, which may happen as a result of a completion of most of the pixel threads or a shift in priority such that the vertex threads now have the highest priority, and then use the Register File 350 to perform vertex calculation operations.

RE claim 4, Lindholm describes a unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations (see the corresponding section in the rejection of claim 3); and

shared resources, operatively coupled to the processor unit (see the corresponding section in the rejection of claim 3);

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough

shared resources become available and then use the shared resources to perform pixel calculation operations (7:37-43, all types of processing threads can use the Register File 350, where thread types include vertex and pixel threads (see 6:43-44).

7:6-36 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, when vertex threads have priority over pixel threads the processor unit will allocate the vertex data to the Register File 350 and will perform vertex calculation operations until enough shared resources become available in the Register File 350 to begin carrying out pixel threads, which may happen as a result of a completion of most of the vertex threads or a shift in priority such that the pixel threads now have the highest priority, and then use the Register File 350 to perform pixel calculation operations.

RE claim 5, Lindholm describes a unified shader comprising:

a processor unit (FIG. 4 “Execution Unit 470” and “PCU 375”);

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store (

8:33-9:32 “Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350."

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

7:6-10 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities".

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, the Scheduler 430 and Instruction Dispatcher 440 are collectively considered a sequencer coupled to the Execution Unit 470, the sequencer maintaining instructions operative to cause the Execution Unit 470 to execute vertex calculation and pixel calculation operations on selected data maintained in a Register File 350 depending upon an amount of space available in the Register File 350).

RE claim 6, Lindholm describes the shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory (

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350”).

RE claim 7, Lindholm describes the shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal (

6:60-7:36 “Thread allocation priority, as described further herein, is used to assign a thread to a source sample. A thread allocation priority is specified for each sample type and Thread Control Unit 420 is configured to assign threads to samples or allocate locations in a Register File 350 based on the priority assigned to each sample type. The thread allocation priority may be fixed, programmable, or dynamic.”

The Thread Control Unit 420 is considered a selection circuit operative to provide information to the store (Register File 350) in response to a control signal, where the control signal is the thread allocation priority associated with each thread or thread type).

RE claim 8, Lindholm describes the shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs (

5:11-35 “Pixel Input Buffer 215 outputs the samples to each Execution Pipeline 240 … Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted ... programmable computation units (PCUs) within an Execution Pipeline 240 ... perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like. Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270.").

RE claim 9, Lindholm describes the shader of claim 5, wherein the processor unit executes vertex calculations while the pixel calculations are still in progress (

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:39-49 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”).

RE claim 10, Lindholm describes the shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs (

4:42-5:35 “Execution Pipelines 240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates … Execution Pipelines 240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 260”).

RE claim 11, Lindholm describes the shader of claim 7, wherein the control signal is provided by an arbiter (

6:60-7:36 “Thread allocation priority, as described further herein, is used to assign a thread to a source sample. A thread allocation priority is specified for each sample type and Thread Control Unit 420 is configured to assign threads to samples or allocate locations in a Register File 350 based on the priority assigned to each sample type. The thread allocation priority may be fixed, programmable, or dynamic … In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220 ... In a further alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on graphics primitive size”.

Thus, while an arbiter isn't explicitly described, the Examiner considers it inherent that some portion of the system acts as an arbiter, and therefore can be considered an arbiter, as some portion of the system assigns priorities to thread and sample types according to the current processing circumstances, in order to more efficiently process the data).

RE claim 12, Lindholm describes a graphics processor comprising:

a unified shader comprising a processor unit that executes vertex calculations while the pixel calculations are still in progress (

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:39-49 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”).

RE claim 13, Lindholm describes the graphics processor of claim 12 wherein the unified shader comprises a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store (see the corresponding section in the rejection of claim 5).

RE claim 14, Lindholm describes the graphics processor of claim 12 comprising a vertex block operative to fetch vertex information from memory (see the rejection of claim 6).

RE claim 15, Lindholm describes a unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload (

7:6-36 “Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220 … In a further alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations

in Register File 350 using thread allocation priorities based on graphics primitive size (number of pixels or fragments included in a primitive)".

9:39-49 "Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.").

RE claim 16, Lindholm describes the shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store (FIG. 4 and 8:15-46 describes Instruction Cache 410, which is considered an instruction store.

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations… Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types."

Thus, the Execution Unit 470 performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store).

Conclusion

THIS ACTION IS MADE FINAL. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not

mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DANIEL WASHBURN whose telephone number is (571)272-5551. The examiner can normally be reached on 9:30 A.M. to 6 P.M..

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ulka Chauhan can be reached on 571-272-7782. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/DANIEL WASHBURN/
Primary Examiner, Art Unit 2628
3/11/12

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	DANIEL WASHBURN	2628

SEARCHED			
Class	Subclass	Date	Examiner
345	501	7/12/11	DW
above	updated	3/11/12	DW

SEARCH NOTES		
Search Notes	Date	Examiner
Searched EAST (all databases) see search history printout	7/12/11	DW
Also see search histories for apps 12/791,597 and 11/842,256	7/12/11	DW
conducted inventor name search	7/12/11	DW
updated search in EAST (all databases) see search history printout	3/11/12	DW

INTERFERENCE SEARCH			
Class	Subclass	Date	Examiner

	/DANIEL WASHBURN/ Primary Examiner.Art Unit 2628

PTO/SB/08a (01-10)
Approved for use through 07/31/2012. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Not for submission under 37 CFR 1.99)

Application Number	13109738
Filing Date	2011-05-17
First Named Inventor	Stephen Morein
Art Unit	2628
Examiner Name	na
Attorney Docket Number	00100.36.0001

U.S.PATENTS Remove

Examiner Initial*	Cite No	Patent Number	Kind Code[1]	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	5550962		1996-08-27	Nakamura et al.	
	2	5818469		1998-10-06	Lawless et al.	
	3	6118452		2000-09-12	Gannett	
	4	6353439		2002-03-05	Lindholm et al.	
	5	6384824		2002-05-07	Morgan et al.	
	6	6417858		2002-07-09	Bosch et al.	
	7	6573893		2003-06-03	Naqvi et al.	
	8	6650327		2002-11-18	Airey et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.W./

Receipt date: 07/14/2011	Application Number	13109738 13109738 - GAU: 2628
INFORMATION DISCLOSURE STATEMENT BY APPLICANT (**Not for submission under 37 CFR 1.99)**	Filing Date	2011-05-17
	First Named Inventor	Stephen Morein
	Art Unit	2628
	Examiner Name	na
	Attorney Docket Number	00100.36.0001

	9	6650330		2003-11-18	Lindholm et al.	
	10	6704018		2004-03-09	Mori et al.	
	11	6724394		2004-04-20	Zatz et al.	
	12	6731289		2004-05-04	Peercy et al.	
	13	6809732		2004-10-26	Zatz et al.	
	14	6864893		2005-03-08	Zatz	
	15	6897871		2005-05-24	Morein et al.	
	16	6980209		2005-12-27	Donham et al.	
	17	7015913		2006-03-21	Lindholm et al.	
	18	7038685		2006-05-02	Lindholm	
	19	7327369		2008-02-05	Morein et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.W./

Receipt date: 07/14/2011	Application Number	13109738	13109738 - GAU: 2628
INFORMATION DISCLOSURE STATEMENT BY APPLICANT (**Not for submission under 37 CFR 1.99)**	Filing Date	2011-05-17	
	First Named Inventor	Stephen Morein	
	Art Unit	2628	
	Examiner Name	na	
	Attorney Docket Number	00100.36.0001	

	20	5485559		1996-01-16	Sakaibara et al.	
	21	7239322	B2	2007-07-03	Lefebvre et al.	
	22	7746348	B2	2010-06-29	Lefebvre et al.	
	23	7742053	B2	2010-06-22	Lefebvre et al.	

If you wish to add additional U.S. Patent citation information please click the Add button. Add

U.S.PATENT APPLICATION PUBLICATIONS Remove

Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	20030076320	A1	2003-04-24	Collodi	
	2	20030164830	A1	2003-09-04	Kent	
	3	20040041814	A1	2004-03-04	Wyatt et al.	
	4	20040164987	A1	2004-08-26	Aronson et al.	
	5	20050068325	A1	2005-03-31	Lefebvre et al.	

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.W./

Receipt date: 07/14/2011	Application Number	13109738	13109738 - GAU: 2628
INFORMATION DISCLOSURE STATEMENT BY APPLICANT (**Not for submission under 37 CFR 1.99)**	Filing Date	2011-05-17	
	First Named Inventor	Stephen Morein	
	Art Unit	2628	
	Examiner Name	na	
	Attorney Docket Number	00100.36.0001	

	6	20100231592	A1	2010-09-16	Morein et al.	
	7	20030030643	A1	2003-02-13	Taylor et al.	
	8	20070222785	A1	2007-09-27	Lefebvre et al.	
	9	20070222787	A1	2007-09-27	Lefebvre et al.	
	10	20050200629	A1	2005-09-15	Morein et al.	
	11	20070222786	A1	2007-09-27	Lefebvre et al.	
	12	20070285427	A1	2007-12-13	Morein et al.	
	13	20100156915	A1	2010-06-24	Lefebvre et al.	

If you wish to add additional U.S. Published Application citation information please click the Add button. Add

FOREIGN PATENT DOCUMENTS Remove

Examiner Initial*	Cite No	Foreign Document Number[3]	Country Code[2] i	Kind Code[4]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear	T[5]
	1	2299408	EP	A2	2011-03-23	Morein et al.		☐

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.W./

Receipt date: 07/14/2011	Application Number	13109738	13109738 - GAU: 2628
INFORMATION DISCLOSURE STATEMENT BY APPLICANT (**Not for submission under 37 CFR 1.99)**	Filing Date	2011-05-17	
	First Named Inventor	Stephen Morein	
	Art Unit	2628	
	Examiner Name	na	
	Attorney Docket Number	00100.36.0001	

	2	2309460	EP	A1	2011-04-13	Morein et al.		☐
	3	2296116	EP	A2	2011-03-16	Morein et al.		☐

If you wish to add additional Foreign Patent Document citation information please click the Add button **Add**

NON-PATENT LITERATURE DOCUMENTS **Remove**

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T[5]
	1	European Patent Office Examination Report; EP Application No. 04798938.9; dated November 9, 2006; pages 1-3.	☐
	2	PURCELL, TIMOTHY J. et al.; Ray Tracing on Programmable Graphics Hardware; SIGGRAPH '02; San Antonio, TX; ACM Transactions on Graphics; July 2002; vol. 21, no. 3; pgs. 703-712.	☐
	3	MARK, WILLIAM R. et al.; CG: A system for programming graphics hardware in a C-like language; SIGGRAPH '03; San Diego, CA; ACM Transactions on Graphics; July 2002; vol. 22, no. 3; pgs. 896-907.	☐
	4	BRETERNITZ, JR., MAURICIO et al.; Compilation, Architectural Support, and Evaluation of SIMD Graphics Pipeline Programs on a General-Purpose CPU; IEEE; 2003; pgs. 1-11.	☐
	5	International Search Report and Written Opinion; International Application No. PCT/IB2004/003821; dated March 22, 2005.	☐
	6	EP Supplemental Search Report; EP Application No. 10075688.1; dated February 25, 2011.	☐
	7	EP Supplemental Search Report; EP Application No. 10075686.5; dated February 25, 2011.	☐

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.W./

Receipt date: 07/14/2011	Application Number	13109738	13109738 - GAU: 2628
INFORMATION DISCLOSURE STATEMENT BY APPLICANT (**Not for submission under 37 CFR 1.99)**	Filing Date	2011-05-17	
	First Named Inventor	Stephen Morein	
	Art Unit	2628	
	Examiner Name	na	
	Attorney Docket Number	00100.36.0001	

	8	EP Supplemental Search Report; EP Application No. 10075687.3; dated February 25, 2011.	☐
	9	EP Supplemental Search Report; EP Application No. 10075685.7; dated February 25, 2011.	☐
	10	ELDRIDGE, MATTHEW et al.; Pomegranate: A Fully Scalable Graphics Architecture; Computer Graphics, SIGGRAPH 2000 Conference Proceedings; July 23, 2000.	☐
	11	OWENS, JOHN D. et al.; Polygon Rendering on a Stream Architecture; Proceedings 2000 SIGGRAPH/Eurographics Workshop on Graphics Hardware; August 21, 2000.	☐
	12	Chinese Office Action; Chinese Application No. 2004800405708; dated September, 2008.	☐
	13	Chinese Office Action; Chinese Application No. 2004800405708; dated November, 2009.	☐
	14	Chinese Office Action; Chinese Application No. 2004800405708; dated September, 2010	☐

If you wish to add additional non-patent literature document citation information please click the Add button Add

EXAMINER SIGNATURE

Examiner Signature	/Daniel Washburn/	Date Considered	03/11/2012

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. [2] Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). [3] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [4] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [5] Applicant is to place a check mark here if English language translation is attached.

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.W./

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	1297	345/501.ccls.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:01
L2	198	lindholm.in. and nvidia.as.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:03
L3	67	lindholm.in. and nvidia.as. and shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:03
L4	45	lindholm.in. and nvidia.as. and shader and vertex and pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:03
L5	2	lindholm.in. and nvidia.as. and shader and vertex and pixel and sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:05
L6	40	lindholm.in. and nvidia.as. and shader and vertex and pixel and sequenc$3	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:05
L7	26	pixel adj input adj buffer and vertex adj input adj buffer and vertex adj output adj buffer and pixel adj output adj buffer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/11 14:29
L8	25	pixel adj input adj buffer and vertex adj	US-PGPUB;	OR	ON	2012/03/11

		input adj buffer and vertex adj output adj buffer and pixel adj output adj buffer and raster adj unit	USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB			14:29
S79	2	10/609967	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/03/09 20:08
S80	36	("20030030643" | "20030076320" | "20030164830" | "20040041814" | "20040164987" | "20050068325" | "20050200629" | "20070222785" | "20070222786" | "20070222787" | "20070285427" | "20100156915" | "20100231592" | "5485559" | "5550962" | "5818469" | "6118452" | "6353439" | "6384824" | "6417858" | "6573893" | "6650327" | "6650330" | "6704018" | "6724394" | "6731289" | "6809732" | "6864893" | "6897871" | "6980209" | "7015913" | "7038685" | "7239322" | "7327369" | "7742053" | "7746348").PN.	US-PGPUB; USPAT	OR	ON	2012/03/11 12:37

EAST Search History (Interference)

<This search history is empty>

3/11/2012 2:55:21 PM
C:\Users\dwashburn1\Documents\EAST\Workspaces\Morein et al. 11117863.wsp

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	DANIEL WASHBURN	2628

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	07/12/2011	03/11/2012							
	1	✓	✓							
	2	✓	✓							
	3	✓	✓							
	4	✓	✓							
	5	✓	✓							
	6	✓	✓							
	7	✓	✓							
	8	✓	✓							
	9	✓	✓							
	10	✓	✓							
	11	✓	✓							
	12	✓	✓							
	13	✓	✓							
	14	✓	✓							
	15	✓	✓							
	16	✓	✓							

PTO/SB/30 (07-09)
Approved for use through 07/31/2012. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Request for Continued Examination (RCE) Transmittal

Address to:
Mail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Application Number	13/109,738
Filing Date	May 17, 2011
First Named Inventor	Stephen Morein
Art Unit	2628
Examiner Name	Daniel C. Washburn
Attorney Docket Number	00100.36.0001

This is a Request for Continued Examination (RCE) under 37 CFR 1.114 of the above-identified application.
Request for Continued Examination (RCE) practice under 37 CFR 1.114 does not apply to any utility or plant application filed prior to June 8, 1995, or to any design application. See Instruction Sheet for RCEs (not to be submitted to the USPTO) on page 2.

1. Submission required under 37 CFR 1.114 Note: If the RCE is proper, any previously filed unentered amendments and amendments enclosed with the RCE will be entered in the order in which they were filed unless applicant instructs otherwise. If applicant does not wish to have any previously filed unentered amendment(s) entered, applicant must request non-entry of such amendment(s).

   a. ☐ Previously submitted. If a final Office action is outstanding, any amendments filed after the final Office action may be considered as a submission even if this box is not checked.

      i. ☐ Consider the arguments in the Appeal Brief or Reply Brief previously filed on _____

      ii. ☐ Other _____

   b. ☑ Enclosed

      i. ☑ Amendment/Reply

      ii. ☐ Affidavit(s)/ Declaration(s)

      iii. ☐ Information Disclosure Statement (IDS)

      iv. ☑ Other Replacement Abstract

2. Miscellaneous

   a. ☐ Suspension of action on the above-identified application is requested under 37 CFR 1.103(c) for a period of _____ months. (Period of suspension shall not exceed 3 months; Fee under 37 CFR 1.17(i) required)

   b. ☐ Other _____

3. Fees The RCE fee under 37 CFR 1.17(e) is required by 37 CFR 1.114 when the RCE is filed.

   a. ☑ The Director is hereby authorized to charge the following fees, any underpayment of fees, or credit any overpayments, to Deposit Account No. 02-0390.

      i. ☑ RCE fee required under 37 CFR 1.17(e)

      ii. ☑ Extension of time fee (37 CFR 1.136 and 1.17)

      iii. ☐ Other _____

   b. ☐ Check in the amount of $ _____ enclosed

   c. ☐ Payment by credit card (Form PTO-2038 enclosed)

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT REQUIRED

Signature	/Christopher J. Reckamp/	Date	September 17, 2012
Name (Print/Type)	Christopher J. Reckamp	Registration No.	34414

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop RCE, Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313-1450 or facsimile transmitted to the U.S. Patent and Trademark Office on the date shown below.

Signature			
Name (Print/Type)		Date	

This collection of information is required by 37 CFR 1.114. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Mail Stop RCE, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**
If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Electronic Patent Application Fee Transmittal

Application Number:	13109738
Filing Date:	17-May-2011
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Filer:	Christopher J. Reckamp/Lisa Schodrowski
Attorney Docket Number:	00100.36.0001

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Extension - 3 months with $0 paid	1253	1	1270	1270

TCL 1002

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
Request for continued examination	1801	1	930	930
		Total in USD ($)		**2200**

Electronic Acknowledgement Receipt

EFS ID:	13761569
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp/Lisa Schodrowski
Filer Authorized By:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001
Receipt Date:	17-SEP-2012
Filing Date:	17-MAY-2011
Time Stamp:	11:54:06
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$2200
RAM confirmation Number	14512
Deposit Account	020390
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

TCL 1002

1	Extension of Time	36001-Extension-Time.pdf	39975 79ce6d6cd14bdf5f5e61865d262cbe8d115a1840	no	1
Warnings:					
Information:					
2		360001-Response.pdf	82375 88229b33dcb3cfc0e421a2497cbf0a02cd9d52dc	yes	11

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Amendment Submitted/Entered with Filing of CPA/RCE	1	1
	Specification	2	3
	Claims	4	7
	Amendment Submitted/Entered with Filing of CPA/RCE	8	9
	Abstract	10	11

Warnings:					
Information:					
3	Request for Continued Examination (RCE)	RCE.pdf	48665 2b5a0b84836700326a0625e81e8130d639efc66b	no	1
Warnings:					
This is not a USPTO supplied RCE SB30 form.					
Information:					
4	Fee Worksheet (SB06)	fee-info.pdf	32304 fc31d73f740d0f1ecad2a9095838aa7f6d88ada4	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	203319		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/22 (09-11)
Approved for use through 07/31/2012. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARMENT OF COMMERCE
Under the paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a)	Docket Number (Optional) 00100.36.0001
Application Number 13/109,738	Filed May 17, 2011
For GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER	
Art Unit 2628	Examiner Daniel C. Washburn

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above identified application.

The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):

		Fee	Small Entity Fee	
☐	One month (37 CFR 1.17(a)(1))	$150	$75	$
☐	Two months (37 CFR 1.17(a)(2))	$560	$280	$
☑	Three months (37 CFR 1.17(a)(3))	$1270	$635	$ 1270.00
☐	Four months (37 CFR 1.17(a)(4))	$1980	$990	$
☐	Five months (37 CFR 1.17(a)(5))	$2690	$1345	$

☐ Applicant claims small entity status. See 37 CFR 1.27.

☐ A check in the amount of the fee is enclosed.

☐ Payment by credit card. Form PTO-2038 is attached.

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☑ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number 02-0390.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

I am the

☐ applicant/inventor.

☐ assignee of record of the entire interest. See 37 CFR 3.71.
Statement under 37 CFR 3.73(b) is enclosed (Form PTO/SB/96).

☑ attorney or agent of record. Registration Number 34414

☐ attorney or agent under 37 CFR 1.34.
Registration number if acting under 37 CFR 1.34 _____

/Christopher J. Reckamp/ — Signature

September 17, 2012 — Date

Christopher J. Reckamp — Typed or printed name

312 356 5094 — Telephone Number

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below.

☑ Total of 1 forms are submitted.

This collection of information is required by 37 CFR 1.136(a). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 6 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

TCL 1002

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Stephen Morein et al.	Examiner:	Daniel C. Washburn
Serial No.:	13/109,738	Art Unit:	2628
Filing Date:	May 17, 2011	Docket No.:	00100.36.0001

Confirmation No.: 2020

Title: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

PRELIMINARY AMENDMENT

Dear Sir:

In response to the final office action mailed March 15, 2012, Applicants submit a Request for Continued Examination, petition for a three month extension of time and submit the following preliminary amendment:

Amendments to the Abstract begin on page 2 of this paper and include a replacement Abstract and a clean copy showing the amended Abstract.

Amendments to the Claims begins on page 3 of this paper.

Remarks begin on page 7 of this paper.

Amendments to the Specification

Please replace the Abstract with the following amended Abstract:

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

A graphics processing architecture in one example performs vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continues pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available. In one example, a general purpose register block maintains data. A sequencer, coupled to the general purpose register block and to a processor unit, maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs. ~~employing a single shader is disclosed. The architecture includes a circuit operative to select one of a plurality of inputs in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the~~

~~plurality of inputs. The shader includes a register block which is used to store the plurality of selected inputs, a sequencer which maintains vertex manipulation and pixel manipulations instructions and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs in response to the instructions maintained in the sequencer.~~

Amendments to the Claims:

Rewrite the claims as set forth below. This listing of claims replaces all prior versions and listings of claims in the application:

Listing of the Claims:

1. (original) A method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

2. (original) A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.

3. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

4. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

5. (original) A unified shader comprising:

a processor unit;

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

6. (original) The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.

7. (original) The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.

8. (original) The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs.

9. (canceled)

10. (original) The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.

11. (original) The shader of claim 7, wherein the control signal is provided by an arbiter.

12. – 14. (canceled)

15. (original) A unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.

16. (original) The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

REMARKS

Applicants respectfully traverse and request reconsideration.

Applicants' attorney wishes to thank the Examiner for the courtesies extended during the telephone conference of September 17, 2012.

Applicants cancel claims 9 and 12-14 without prejudice. Applicants have also amended the Abstract.

Claims 1-16 stand rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by U.S. Patent No. 7,038,685 (Lindholm). Applicants respectfully request reconsideration and respectfully submit that the declaration is proper and that the declaration is more than "vague general statements in broad terms…". To the contrary, the statements and Exhibits set for facts sufficient to show a conception and reduction to practice sufficient to show priority of invention. To the extent additional information would be helpful, Applicants respectfully submit by way of example that:

As to claim 1 for example, Exhibit B Chip Design Code – sq_gpr_alloc.v and Sq_alu_instr_seq.v – are believed to illustrate, inter alia, loading either pixel or vertices in the GPR if there is space for them (e.g., transmission to general purpose register (gpr) blocks unless the gpr block does not have space); performing pixel and vertex manipulations; the ais machine is the "alu instruction sequencer" and it executes instructions on either vertices or pixels depending on type. the file sq_instruction_store.v contains the memory with the instructions to be performed on either pixels (PS) or vertices (VS).

As to claims 2-5 for example, Exhibit B Chip Design Code – sp_macc_gpr.v, SP_vector.v, Sq.v , Sq_export_alloc.v, sq_ctl_flow_seq.v, Sq_alu_instr_seq.v - are believed to illustrate, inter alia, the general purpose register and processor (e.g., multiply and accumulate

(MAC or MACC) logic) and a sequencer coupled to the general purpose register and processor unit and operation of the sequencer and processor unit.

Applicant respectfully submits that the claims are now believed to be in condition for allowance and that a timely Notice of Allowance be issued in this case. If the Examiner believes that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at (312) 356-5094.

Respectfully submitted,

Dated: September 17, 2012

By: /Christopher J. Reckamp/
Christopher J. Reckamp
Reg. No. 34,414

Faegre Baker Daniels LLP
311 S. Wacker Drive
Chicago, IL 60606
PHONE: (312) 356-5094
FAX: (312) 212-6501

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

A graphics processing architecture in one example performs vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continues pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available. In one example, a general purpose register block maintains data. A sequencer, coupled to the general purpose register block and to a processor unit, maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

A graphics processing architecture in one example performs vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continues pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available. In one example, a general purpose register block maintains data. A sequencer, coupled to the general purpose register block and to a processor unit, maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	☐ To be Mailed
13/109,738	05/17/2011	

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY ☐ RATE ($)	SMALL ENTITY ☐ FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT	09/17/2012	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	* 12	Minus	** 20	= 0	X $ =		OR	X $60=	0
	Independent (37 CFR 1.16(h))	* 6	Minus	***6	= 0	X $ =		OR	X $250=	0
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	0

AMENDMENT		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.

** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".

*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/LAWANDA MILTON/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

29153 7590 12/06/2012

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

EXAMINER	
CHEN, FRANK S	

ART UNIT	PAPER NUMBER
2677	

NOTIFICATION DATE	DELIVERY MODE
12/06/2012	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

inteas@faegrebd.com
cynthia.payson@faegredb-.com
michelle.davis@faegrebd.com

Office Action Summary	Application No.	Applicant(s)	
	13/109,738	MOREIN ET AL.	
	Examiner	Art Unit	
	FRANK CHEN	2677	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 September 2012*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

5)☒ Claim(s) *1-8,10,11,15 and 16* is/are pending in the application.

5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6)☐ Claim(s) _____ is/are allowed.

7)☒ Claim(s) *1-8,10,11,15 and 16* is/are rejected.

8)☐ Claim(s) _____ is/are objected to.

9)☐ Claim(s) _____ are subject to restriction and/or election requirement.

* If any claims have been determined allowable, you may be eligible to benefit from the **Patent Prosecution Highway** program at a participating intellectual property office for the corresponding application. For more information, please see http://www.uspto.gov/patents/init_events/pph/index.jsp or send an inquiry to PPHfeedback@uspto.gov.

Application Papers

10)☐ The specification is objected to by the Examiner.

11)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

3) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

4) ☐ Other: _____.

DETAILED ACTION

Claim Status

1. Claims 1-8,10-11, and 15-16 are currently pending in this application.

2. Claims 9 and 12-14 have been canceled.

Specification

3. Applicant is reminded of the proper language and format for an abstract of the disclosure.

The abstract should be in narrative form and **generally limited to a single paragraph on a separate sheet within the range of 50 to 150 words**. The form and legal phraseology often used in patent claims, such as "means" and "said," should be avoided. The abstract should describe the disclosure sufficiently to assist readers in deciding whether there is a need for consulting the full patent text for details.

The language should be clear and concise and should not repeat information given in the title. It should avoid using phrases which can be implied, such as, "The disclosure concerns," "The disclosure defined by this invention," "The disclosure describes," etc.

4. The abstract of the disclosure is objected to because it exceeds 150

words. Correction is required. See MPEP § 608.01(b).

Declaration filed under 37 CFR 1.131

The declaration filed 1/18/12 under 37 CFR 1.131 and the Applicants

Argument/Remarks Made in an Amendment filed 9/17/2012 have been

considered but is ineffective to overcome the prior art reference Lindholm (US

7,038,685, “the Lindholm reference”).

The declaration does not meet the requirements of 37 CFR 1.131 section

(a).

37 CFR 1.131 section (a) states (in relevant part):

"(a) When any claim of an application or a patent under reexamination is rejected, the inventor of the subject matter of the rejected claim, the owner of the patent
under reexamination, or the party qualified under §§ 1.42, 1.43, or 1.47, may submit an appropriate oath or declaration to establish invention of the subject matter of the rejected claim prior to the effective date of the reference or activity on which the rejection is based. The effective date of a U.S. patent, U.S. patent application publication, or international application publication under PCT Article 21(2) is the earlier of its publication date or date that it is effective as a reference under 35 U.S.C. 102(e). **Prior invention may not be established under this section in any country other than the United States, a NAFTA country, or a WTO member country. Prior invention may not be established under this section before December 8, 1993, in a NAFTA country other than the United States, or before January 1, 1996, in a WTO member country other than a NAFTA country.**" (emphasis added)

Section 2 of Applicants' declaration describes (in relevant part):

"2. We conceived the Invention prior to June 30, 2003 while employed by ATI Technologies Inc. and/or one of its wholly owned subsidiaries ("ATI") as indicated by attached Exhibits A and B ... Prior to June 30, 2003 we created a graphics processing system that operated as claimed using a computer system that successfully executed the Model Code. Prior to June 30, 2003 we also created a graphics processing system as claimed in the form of a computer system that used an RTL simulator to successfully validate the operation of an integrated circuit version of the claimed graphics processing system and method."

As quoted from Applicants' declaration, section 2 describes conception and reduction to practice of the claimed invention prior to June 30, 2003. Section 2 further describes that the conception and reduction to practice of the claimed invention was carried out while the inventors were employed by ATI Technologies Inc. and/or one of its wholly owned subsidiaries.

However, section 2, and the declaration as a whole, fails to specify whether or not the conception and reduction to practice was carried out in the United States, a NAFTA country, or a WTO member country. As quoted from 37 CFR 1.131 section (a), "[p]rior invention may not be established under this

section in any country other than the United States, a NAFTA country, or a WTO memory country". Thus, the declaration is ineffective to overcome the Lindholm reference due to this first deficiency.

Moreover, the applicants in their Remarks filed on 9/17/2012 do not appear to address this issue. In the Remarks, the applicants attempt to further correlate the claim limitations to the submitted reduction to practice evidence (Exhibit B Chip Design Code) of the Declaration Under 37 CFR 1.131 filed on 1/18/2012 but do not appear to show that reduction to practice was carried out in the United States, a NAFTA country, or a WTO member country. Therefore, the declaration continues to not meet the requirements of 37 CFR 1.131 section (a).

Further, the declaration does not meet the requirements of 37 CFR 1.131 section (b).

37 CFR 1.131 section (b) states:

"(b) The showing of facts shall be such, in character and weight, as to establish reduction to practice prior to the effective date of the reference, or conception of the invention prior to the effective date of the reference coupled with due diligence from prior to said date to a subsequent reduction to practice or to the filing of the application. Original exhibits of drawings or records, or photocopies thereof, must accompany and form part of the affidavit or declaration or their absence must be satisfactorily explained."

MPEP 715.07 [R-3] "Facts and Documentary Evidence", section I. "General Requirements", offers further guidance regarding the requirements of 37 CFR 1.131 section (b).

MPEP 715.07, section I., describes (in relevant part):

"The essential thing to be shown under 37 CFR 1.131 is priority of invention and this may be done by any satisfactory evidence of the fact. FACTS, not conclusions, must be alleged. Evidence in the form of exhibits may

accompany the affidavit or declaration. Each exhibit relied upon should be specifically referred to in the affidavit or declaration, in terms of what it is relied upon to show … when reviewing a 37 CFR 1.131 affidavit or declaration, the examiner must consider all of the evidence presented in its entirety, including the affidavits or declarations and all accompanying exhibits, records and "notes." An accompanying exhibit need not support all claimed limitations, provided that any missing limitation is supported by the declaration itself. Ex parte Ovshinsky, 10 USPQ2d 1075 (Bd. Pat. App. & Inter. 1989).

The affidavit or declaration and exhibits must clearly explain which facts or data applicant is relying on to show completion of his or her invention prior to the particular date. Vague and general statements in broad terms about what the exhibits describe along with a general assertion that the exhibits describe a reduction to practice "amounts essentially to mere pleading, unsupported by proof or a showing of facts" and, thus, does not satisfy the requirements of 37 CFR 1.131(b). In re Borkowski, 505 F.2d 713, 184 USPQ 29 (CCPA 1974). **Applicant must give a clear explanation of the exhibits pointing out exactly what facts are established and relied on by applicant.** 505 F.2d at 718-19, 184 USPQ at 33. See also In re Harry, 333 F.2d 920, 142 USPQ 164 (CCPA 1964) (Affidavit "asserts that facts exist but does not tell what they are or when they occurred.")." (emphasis added)

Applicants' Remarks filed on 09/17/2012 contains the following in the second to last paragraph which recites:

"As to claims 2-5 for example, Exhibit B Chip Design Code - p_macc_gpr.v, SP_vector.v, Sq.v, Sq_export_alloc.v, sq ctl flow_seq.v, Sq_alu_instr_seq.v - are believed to illustrate, inter alia, the general purpose register and processor (e.g., multiply and accumulate (MAC or MACC) logic) and a sequencer coupled to the general purpose register and processor unit and operation of the sequencer and processor unit."

However, this paragraph as a whole is considered nothing more than vague and general statements in broad terms about what the exhibits describe along with general assertions that the exhibits describe a reduction to practice, which does not satisfy the requirements of 37 CFR 1.131 section (b). Thus, the declaration in view of the Remark is ineffective to overcome the Lindholm reference due to this second deficiency.

Regarding claim 1, the Examiner is able to determine which sections of Exhibit B Chip Design Code corresponds to which limitations of Claim 1 after reviewing the Remarks filed on 9/17/2012 . However, the Examiner is unable to do so for Claims 2-5 as they are not satisfactorily explained in the Remarks.

Therefore, the most recent declaration filed 1/18/12 under 37 CFR 1.131 and Remarks filed on 9/17/2012 are together ineffective to overcome the Lindholm reference. As an additional note, the Examiner would like to point out that US Pat 7,015,913, to Lindholm et al., filed June 27th, 2003, appears, after brief review, to include a disclosure that is similar to US Pat 7,038,685 to Lindholm, which is used in the rejections that follow (see FIG. 2 of each patent). The Examiner has not given Lindholm et al. (US 7,015,913) a thorough review as to whether or not it teaches one or more of Applicants' claims, but it may be worth Applicants' time to review Lindholm et al. (US 7,015,913) and adjust the declaration such that conception and reduction to practice of the claimed invention is declared to have occurred prior to June 27th, 2003 (if such a statement is true), in order to avoid a future rejection based on the teachings of prior art reference Lindholm et al. (US 7,015,913).

Claim Rejections - 35 USC § 112

5. The following is a quotation of 35 U.S.C. 112(b):

(B) CONCLUSION.—The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the inventor or a joint inventor regards as the invention.

The following is a quotation of 35 U.S.C. 112 (pre-AIA), second paragraph:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

6. Claim 2 recites the limitation "the selected one of the plurality of inputs" in "in response to the selected one of the plurality of inputs." There is insufficient antecedent basis for this limitation in the claim. Proper amendment is requested.

7. Claim 8 recites the limitation "the selected one of the plurality of inputs" in "in response to the selected one of the plurality of inputs." There is insufficient antecedent basis for this limitation in Claim 8 or Claim 5. Proper amendment is requested.

Claim Rejections - 35 USC § 102

8. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-8, 10-11, and 15-16 are rejected under 35 U.S.C. 102(e) as being anticipated by Lindholm (US 7,038,685).

RE claim 1, Lindholm describes a method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the

general purpose register block does not have enough available space therein to

store incoming vertex data (

3:59-65: “Programmable Graphics Processing Pipeline 150 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample or any other data. For simplicity, the remainder of this description will use the term 'samples' to refer to graphics data such as surfaces, primitives, vertices, pixels, fragments, or the like."

6:38-59: “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

7:6-10: “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities".

7:36-43: “Once a thread is assigned to a source sample, the thread is allocated storage resources such as locations in a Register File 350 to retain intermediate data generated during execution of program instructions associated with the thread."

9:33-56: "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

15:7-13: “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, Lindholm describes performing vertex manipulation operations and

pixel manipulation operations by transmitting vertex data to a general purpose

register block (sample data, such as vertex or pixel data, is transmitted to

Register File 350) and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data (the multi-threaded processing unit 400 carries out vertex operations on vertex data unless the Register File 350 doesn't have enough room to store the incoming vertex data, in which case the thread associated with the vertex data and vertex operations must wait until enough space becomes available); and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available (

7:6-21: "In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

8:15-58: "Thread Selection Unit 415 reads one or more thread entries based on thread execution priorities and outputs selected thread entries to Instruction Cache 410. Instruction cache 410 determines if the program instructions corresponding to the program counters and sample type included in the thread state data for each thread entry are available in Instruction Cache 410 ... The program instructions corresponding to the program counters from the one or more thread entries are output by Instruction Cache 410 to … Instruction Scheduler 430 … Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU [instruction window unit] 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350. An instruction specifies the location of source data needed to execute the instruction."

15:7-13: "In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420

does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, Lindholm is considered to describe an embodiment including continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available, as the Execution Unit 470 may be carrying out calculations for one or more high priority pixel threads based on instructions stored in Instruction Cache 410 and/or IWU 435 while a low priority vertex thread is waiting for the one or more pixel threads to finish such that when the pixel threads finish the system will deallocate the resources assigned to the completed pixel threads in the Register File 350 and will allocate the requested amount of resources to the queued up vertex thread).

RE claim 2, Lindholm describes a unified shader, comprising:

a general purpose register block for maintaining data (

7:37-43: “Once a thread is assigned to a source sample, the thread is allocated storage resources such as locations in a Register File 350 to retain intermediate data generated during execution of program instructions associated with the thread.”);

a processor unit (FIG. 4 “Execution Unit 470” and “PCU 375”);

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block (

8:33-9:32 “Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350."

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."); and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs (

9:39-46 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … and output the processed sample to a destination specified by the instruction. The destination may be Vertex Output Buffer 260, Pixel Output Buffer 270, or Register File 350.”

4:42-5:35 “Execution Pipelines 240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates … Execution Pipelines 240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 260 ... Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted ... programmable computation units (PCUs) within an Execution Pipeline 240 ... perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like. Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270."

Thus, the Execution Unit 470 is considered a processor unit that executes instructions that generate a pixel color in response to the selected one of the

plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs (also see 4:22-5:35)).

RE claim 3, Lindholm describes a unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations (FIG. 4 “Execution Unit 470” and “PCU 375”.

6:38-59 “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multi-Threaded Processing Unit 400 ... In one embodiment TSR [Thread Storage Resource] 325 stores thread data for each of at least two thread types, where the at least two thread types may include pixel, primitive and vertex.”

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

Thus, the Execution Unit 470 and internal PCU 375 are collectively considered a processor unit operative to perform vertex calculation operations and pixel calculation operations); and

shared resources, operatively coupled to the processor unit (FIG. 4 illustrates Register File 350 coupled to Execution Unit 470, and 7:37-43 describes that the Register File 350 is shared among threads);

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations (7:37-43, all types of

processing threads can use the Register File 350, where thread types include vertex and pixel threads (see 6:43-44).

7:6-36 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, when pixel threads have priority over vertex threads the processor unit will allocate the pixel data to the Register File 350 and will perform pixel calculation operations until enough shared resources become available in the Register File 350 to begin carrying out vertex threads, which may happen as a result of a completion of most of the pixel threads or a shift in priority such that the vertex threads now have the highest priority, and then use the Register File 350 to perform vertex calculation operations.

RE claim 4, Lindholm describes a unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations (see the corresponding section in the rejection of claim 3); and

shared resources, operatively coupled to the processor unit (see the corresponding section in the rejection of claim 3);

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations (7:37-43, all types of processing threads can use the Register File 350, where thread types include vertex and pixel threads (see 6:43-44).

7:6-36 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220."

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, when vertex threads have priority over pixel threads the processor unit will allocate the vertex data to the Register File 350 and will perform vertex calculation operations until enough shared resources become available in the Register File 350 to begin carrying out pixel threads, which may happen as a result of a completion of most of the vertex threads or a shift in priority such that the pixel threads now have the highest priority, and then use the Register File 350 to perform pixel calculation operations.

RE claim 5, Lindholm describes a unified shader comprising:

a processor unit (FIG. 4 “Execution Unit 470” and “PCU 375”);

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store (

8:33-9:32 “Each clock cycle, Instruction Scheduler 430 evaluates whether any instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350."

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations."

7:6-10 “In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities".

15:7-13 “In step 877 Thread Control Unit 320 or 420 determines if storage resources for storing intermediate data generated during execution of the thread are available. The storage resources may be in graphics memory. When storage resources are not available in step 877, Thread Control Unit 320 or 420 does not proceed to step 880 until a storage resources become available. In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375."

Thus, the Scheduler 430 and Instruction Dispatcher 440 are collectively considered a sequencer coupled to the Execution Unit 470, the sequencer maintaining instructions operative to cause the Execution Unit 470 to execute vertex calculation and pixel calculation operations on selected data maintained in a Register File 350 depending upon an amount of space available in the Register File 350).

RE claim 6, Lindholm describes the shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory (

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350").

RE claim 7, Lindholm describes the shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal (

6:60-7:36 "Thread allocation priority, as described further herein, is used to assign a thread to a source sample. A thread allocation priority is specified for each sample type and Thread Control Unit 420 is configured to assign threads to samples or allocate locations in a Register File 350 based on the priority assigned to each sample type. The thread allocation priority may be fixed, programmable, or dynamic."

The Thread Control Unit 420 is considered a selection circuit operative to provide information to the store (Register File 350) in response to a control signal, where the control signal is the thread allocation priority associated with each thread or thread type).

RE claim 8, Lindholm describes the shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs (

5:11-35 "Pixel Input Buffer 215 outputs the samples to each Execution Pipeline 240 … Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted ... programmable computation units (PCUs) within an Execution Pipeline 240 ... perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like. Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270.").

RE claim 10, Lindholm describes the shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs (

4:42-5:35 “Execution Pipelines 240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates … Execution Pipelines 240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 260”).

RE claim 11, Lindholm describes the shader of claim 7, wherein the control signal is provided by an arbiter (

6:60-7:36 “Thread allocation priority, as described further herein, is used to assign a thread to a source sample. A thread allocation priority is specified for each sample type and Thread Control Unit 420 is configured to assign threads to samples or allocate locations in a Register File 350 based on the priority assigned to each sample type. The thread allocation priority may be fixed, programmable, or dynamic … In an alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220 ... In a further alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on graphics primitive size”.

Thus, while an arbiter isn't explicitly described, the Examiner considers it inherent that some portion of the system acts as an arbiter, and therefore can be considered an arbiter, as some portion of the system assigns priorities to thread and sample types according to the current processing circumstances, in order to more efficiently process the data).

RE claim 15, Lindholm describes a unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload (

7:6-36 “Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on an amount of sample data in Pixel Input Buffer 215 and another amount of sample data in Vertex Input Buffer 220 … In a further alternate embodiment, Thread Control Unit 420 is configured to assign threads to source samples or allocate locations in Register File 350 using thread allocation priorities based on graphics primitive size (number of pixels or fragments included in a primitive)”.

9:39-49 “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations … Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”).

RE claim 16, Lindholm describes the shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store (FIG. 4 and 8:15-46 describes Instruction Cache 410, which is considered an instruction store.

9:33-56 "Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375 ... Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 to perform operations… Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.”

Thus, the Execution Unit 470 performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store).

9. Additionally, Claims 1-8, 10-11, and 15-16 are further rejected under 35 U.S.C. 103 as being unpatentable over Shen et al (U.S. Patent No. 7,646817B1) in view of Parikh et al. (U.S. Patent No. 6,697,074 B2).

10 Regarding Claim 1, Shen discloses **A method comprising: performing vertex manipulation operations and pixel manipulation operations** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.) **and performing vertex operations on the vertex data by a processor and** (*Col. 4, lines 8-12* reciting "Exemplary GPU 208 includes a programmable vertex shader 212 for performing graphics operations on a per-vertex basis, and a programmable pixel shader 214 for performing graphics operations on a per-pixel basis." The programmable vertex shader performs graphics operations on vertex data sent to it, thus the programmable vertex shader 212 processes vertex

data and the vertex shader 212 is included within the GPU 208, which corresponds to a processor.)

continuing pixel calculation operations that are to be or are currently being performed by the processor (*Col. 6, lines 16* reciting "At block 322, video decoding application 216 directs the pixel shader component 214 of GPU 208 to perform color space conversion processing on the reconstructed image. Color space conversion processing is performed pixel-by-pixel to convert an image from a color space in which it was created (e.g., YUV) to a color space supported by display device 204 (e.g., RGB)." The color space conversion corresponds to pixel calculation operations that are to be performed by the processor because pixel shader is acting on pixel calculations that occur after vertex calculation operations and is performed by the pixel shader component of the GPU 208.) **based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.** (*Col. 4, lines 30-32* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer." Accelerated video decoding which includes per-pixel operations is described in computer-executable instructions. These instructions which are in the form of computer-executable instructions are used for execution. The computer-readable memory medium corresponds to the instruction store that stores the computer-executable instructions.)

While Shen does not explicitly disclose **by transmitting vertex data to a general purpose register block,** and **unless the general purpose register block does not have enough available space therein to store incoming vertex data;** Parikh does disclose **by transmitting vertex data to a general purpose register block,** (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are general purpose registers for storing at least pixel and vertex data and additional data formats.)

unless the general purpose register block does not have enough available space therein to store incoming vertex data; (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." The number of registers available in the graphics processor will be finite and they may all be filled with only pixel format (pixel data). Therefore, if all the registers are all filled with non-vertex data, the processor may not read and process vertex format (vertex data).)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are

processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

11. Regarding Claim 2, Shen discloses **a processor unit;** (*Col. 4, lines 8-12* reciting "Exemplary GPU 208 includes a programmable vertex shader 212 for performing graphics operations on a per-vertex basis, and a programmable pixel shader 214 for performing graphics operations on a per-pixel basis." The GPU 208, which corresponds to a processor.)

a sequencer, coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of

computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The memory storage devices corresponds to the sequencer because it stores the computer-executable instructions, such as application modules, which are in a sequence. The application modules may be located in local computer storage media and such local storage medium is coupled to the processor since it is accessible by the processor.)

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs. (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects

processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to receiving a select input to perform. The special effects processing such as inverse telecine and scalling or fading corresponds to generating pixel color.)

While Shen does not explicitly disclose **A unified shader, comprising: a general purpose register block for maintaining data;** and **general purpose register block and the** Parikh does disclose **A unified shader, comprising: a general purpose register block for maintaining data;** (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are general purpose registers that can store at least pixel and vertex data and additional formats of data.)

general purpose register block and the (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are general purpose registers that can store at least pixel and vertex data and additional formats of data.)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

12. Regarding Claim 3, Shen discloses **A unified shader comprising:**

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based

special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.)

While Shen does not disclose **shared resources, operatively coupled to the processor unit;** Parikh does disclose **shared resources, operatively coupled to the processor unit;** (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are shared resources that may be used to store at least pixel formats, vertex formats, and additional data formats.)

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations. (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to

registers within the graphics and audio processors." The registers available in the graphics processor will be finite and they may all be filled with pixel format (pixel data). Therefore, if there is no empty registers left and the registers are all filled with non-vertex data, the processor may not read and process vertex format (vertex data).)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

13. Regarding Claim 4, Shen discloses **A unified shader comprising: a processor unit operative to perform vertex calculation operations and pixel calculation operations; and** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.)

Parikh discloses **shared resources, operatively coupled to the processor unit;** (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." The registers available in the graphics processor will be finite and they may all be filled with pixel format (pixel data). Therefore, if there is no empty registers left and the registers are all filled with non-vertex data, the processor may not read and process vertex format (vertex data).)

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations. (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex

values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

14. Regarding Claim 5, Shen further discloses **A unified shader comprising: a processor unit;** (*Col. 4, lines 8-12* reciting "Exemplary GPU 208 includes a programmable vertex shader 212 for performing graphics operations on a per-vertex basis, and a programmable pixel shader 214 for performing graphics operations on a per-pixel basis." The GPU 208, which corresponds to a processor.)

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store. (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform

particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The memory storage devices corresponds to the sequencer because it stores the computer-executable instructions, such as application modules, which are in a sequence. The application modules may be located in local computer storage media and such local storage medium is coupled to the processor since it is accessible by the processor.)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the

graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

15. Regarding Claim 6, Shen further discloses **The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.** (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The communications network based on communication protocols

corresponds to circuitry operative to fetch the instructions from the remote computer storage media.)

16. Regarding Claim 7, Shen further discloses **The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.** (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The communications networks also corresponds to selective circuit that provides information to the memory storage devices.)

17. Regarding Claim 8, Shen further discloses **The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For

example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." Fading in or out corresponds to pixel color in response to the GPU 208 receiving directions (plurality of inputs) from the video decoding application 216.)

18. Regarding Claim 10, Shen further discloses **The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." Both scaling and inverse telecine corresponds to vertex position (scaling) and appearance data (reverse telecine).)

19. Regarding Claim 11, Shen further discloses **The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." De-

interlacing, inverse telecine, and scaling all correspond to vertex position (scaling) and appearance data (reverse telecine).)

20. Regarding Claim 15, Shen discloses **A unified shader comprising: a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. The GPU corresponds to a processor unit flexibly controlled, and the reconstructed image corresponds to the workload since the reconstructed image will have varying numbers of vertex and pixel data to process.)

21. Regarding Claim 16, Shen further discloses **The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.** (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of

programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The local and remote computer storage media including memory storage devices corresponds to the instruction store. Computer executable instructions corresponds to the vertex and pixel manipulation operations which is completed at various degrees according to the structure of the application module (stored instructions) on the storage device (instruction store).)

CONTACT

22. Any inquiry concerning this communication or earlier communications from the examiner should be directed to FRANK CHEN whose telephone number is (571)270-7993. The examiner can normally be reached on 8 - 5, Monday - Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kee Tung can be reached on (571)272-7794. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public

PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/FRANK CHEN/
Examiner, Art Unit 2677

/KEE M TUNG/

Supervisory Patent Examiner, Art Unit 2677

Notice of References Cited	Application/Control No. 13/109,738	Applicant(s)/Patent Under Reexamination MOREIN ET AL.	
	Examiner FRANK CHEN	Art Unit 2677	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-6,697,074 B2	02-2004	Parikh et al.	345/522
*	B	US-7,646,817 B2	01-2010	Shen et al.	375/240.25
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	**Art Unit**
	FRANK CHEN	2677

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	07/12/2011	03/11/2012	11/30/2012						
	1	✓	✓	✓						
	2	✓	✓	✓						
	3	✓	✓	✓						
	4	✓	✓	-						
	5	✓	✓	✓						
	6	✓	✓	✓						
	7	✓	✓	✓						
	8	✓	✓	✓						
	9	✓	✓	-						
	10	✓	✓	✓						
	11	✓	✓	✓						
	12	✓	✓	-						
	13	✓	✓	-						
	14	✓	✓	-						
	15	✓	✓	✓						
	16	✓	✓	✓						

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	2	"7646817".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/12/01 14:23
L2	2	"6697074".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/12/01 14:23
S1	108	single WITH shader WITH pixel WITH vertex	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 14:43
S2	94	unified ADJ shader$2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:28
S3	94	unified ADJ shader$2	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:28
S4	131	pixel WITH vertex WITH combination WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:43
S5	400	combination WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:52
S6	32	combination ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:52
S7	868	single WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:47
S8	145	single NEAR shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:47
S9	127	single ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S10	108	single WITH shader WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S11	10	single NEAR shader WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO;	OR	ON	2012/11/29 20:48

TCL 1002

			JPO; DERWENT; IBM_TDB			
S12	145	single NEAR shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:50
S13	0	combin3 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:06
S14	4	integrated WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:20
S15	0	simultaneou$4 WITH pixel WITH vertex WITH sahder	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:24
S16	23	simultaneou$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:24
S17	42	concurrent WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:25
S18	2	coexist WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:26
S19	0	contemporaneous WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:28
S20	1	contemporary WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:29
S21	0	synchron$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:32
S22	9	combined WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:33
S23	0	cumulat$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S24	4	composite WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S25	8	together WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35

S26	44	incorporat$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:38
S27	0	integration WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:43
S28	0	consolida$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S29	8	cooperat$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S30	0	undivided WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S31	381	one WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S32	23	one WITH only WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S33	108	single WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:45
S34	94	unified ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 09:03
S35	6	Lindoln	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S36	5041	Lindholm	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S37	215	Lindholm AND shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S38	138	Lindholm AND programmable WITH graphics WITH processor	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S39	297	graphics WITH processor WITH vertex WITH pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:09
S40	294	processor WITH vertex WITH pixel WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/30 10:48

			IBM_TDB			
S41	131	graphics WITH processor WITH vertex WITH pixel WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:54
S42	104	graphics WITH processor WITH vertex WITH pixel WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:54
S43	105	graphics WITH processor WITH vertex WITH pixel WITH processing	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:00
S44	1	graphics WITH processor WITH vertex WITH pixel WITH processing WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:01
S45	81	graphics WITH processor WITH vertex WITH pixel WITH operations	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:29
S46	0	graphics WITH processor WITH vertex WITH manipulation WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S47	0	processor WITH vertex WITH manipulation WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S48	21	processor WITH vertex WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S49	2	graphics WITH processor WITH dual WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:27
S50	116	GPU WITH vertex WITH pixel WITH (operations OR manipulation OR calculation)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:28
S51	14	vertex WITH data WITH general WITH purpose WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:10
S52	17	vertex WITH data WITH general WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:11
S53	18	general WITH purpose WITH register WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:12
S54	10	((general$1purpose WITH register) OR GPR) WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:15
S55	18	general WITH purpose WITH	US-PGPUB; USPAT;	OR	ON	2012/11/30

		register WITH vertex	USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB			15:19
S56	25	general WITH purpose WITH register SAME vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:19
S57	94	vertex WITH data WITH store WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:56
S58	43	vertex NEAR data WITH store WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:59
S59	121	vertex NEAR data WITH stor$3 WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:22
S60	4	vertex NEAR data WITH transmit WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:43
S61	19	vertex NEAR data WITH transmit$5 WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:43
S62	194	vertex WITH pixel WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 21:28

EAST Search History (Interference)

< This search history is empty>

12/ 1/ 2012 2:40:15 PM
C:\ Users\ fchen\ Documents\ EAST\ Workspaces\ 13109738_20110216077_Morein_et_al.wsp

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	FRANK CHEN	2677

SEARCHED			
Class	Subclass	Date	Examiner
345	501	7/12/11	DW
above	updated	3/11/12	DW

SEARCH NOTES		
Search Notes	Date	Examiner
Searched EAST (all databases) see search history printout	7/12/11	DW
Also see search histories for apps 12/791,597 and 11/842,256	7/12/11	DW
conducted inventor name search	7/12/11	DW
updated search in EAST (all databases) see search history printout	3/11/12	DW
updated search in EAST (all databases) see search history printout	11/30/2012	FC

INTERFERENCE SEARCH			
Class	Subclass	Date	Examiner

/FRANK CHEN/ Examiner.Art Unit 2677	

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Stephen Morein et al.	Examiner:	Frank S. Chen
Serial No.:	13/109,738	Art Unit:	2677
Filing Date:	May 17, 2011	Docket No.:	00100.36.0001

Confirmation No.: 2020

Title: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

AMENDMENT AND RESPONSE

Dear Sir:

In response to the office action mailed December 6, 2012, Applicants petition for a three month extension of time and respond as follows:

Amendments to the Abstract begin on page 2 of this paper and include a replacement Abstract and a clean copy showing the amended Abstract.

Amendments to the Claims begins on page 3 of this paper.

Remarks begin on page 7 of this paper.

Amendments to the Specification

Please replace the Abstract with the following amended Abstract:

ABSTRACT

A graphics processing architecture in one example performs vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continues pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available. ~~In one example, a general purpose register block maintains data. A sequencer, coupled to the general purpose register block and to a processor unit, maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.~~

Amendments to the Claims:

Rewrite the claims as set forth below. This listing of claims replaces all prior versions and listings of claims in the application:

Listing of the Claims:

1. (original) A method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

2. (currently amended) A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and

wherein the processor unit executes instructions that generate a pixel color in response to [[the]] selected data from the general purpose register block ~~one of the plurality of inputs~~ and

generates vertex position and appearance data in response to [[a]] selected data from the general purpose register block. ~~one of the plurality of inputs.~~

3. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

4. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

5. (original) A unified shader comprising:

a processor unit;

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation

operations on selected data maintained in a store depending upon an amount of space available in the store.

6. (original) The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.

7. (original) The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.

8. (currently amended) The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected data. ~~one of the plurality of inputs.~~

9. (canceled)

10. (currently amended) The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected data. ~~one of the plurality of inputs.~~

11. (original) The shader of claim 7, wherein the control signal is provided by an arbiter.

12. – 14. (canceled)

15. (original) A unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.

16. (original) The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

REMARKS

Applicants respectfully traverse and request reconsideration.

The Specification has been objected to. Applicants submit herewith an amended Abstract (including a clean copy of the amended Abstract and replacement sheet).

The declaration filed 1/18/12 under 37 CFR 1.131 and the Applicants Argument/Remarks Made in an Amendment filed 9/17/2012 are allegedly ineffective to overcome the prior art reference Lindholm due to certain portions in the affidavit/exhibits. Applicants respectfully traverse and submit that the Applicants' declaration and exhibits are in proper form. It is alleged that the declaration does not meet requirements of 37 CFR 1.131(a). Applicants respectfully submit that their original declaration indicates that they are citizens of the U.S. or a NAFTA country. Applicants conception reduction of practice was carried out in the United States.

As to the rejection under MPEP 715.07 section I, Applicants respectfully note that the evidence is "in the form of exhibits and may accompany the affidavit or declaration". As stated, "the Examiner must consider all of the evidence presented in its entirety, including the affidavits or declarations and all accompanying exhibits, records and notes" and respectfully submit that the exhibits clearly explain the data Applicants are relying on to show completion of the invention in combination with the explanation provided in the declaration, as understood by one of ordinary skill in the art. Applicants believe that there is a clear explanation in the exhibits as to claims 2-5 pointing out exactly what facts are established and relied upon by Applicants.

For example, on page 4 of the Exhibit B Chip Design Code the words "claim 2:" appear referencing that the following code segments refer to the reduction of practice of claim 2. Applicants also made an attempt to provide additional comments in the exhibits that clearly set forth the proof and showing of facts that are believed to satisfy the requirements of 37 CFR

1.131. Applicants respectfully request reconsideration. Applicants invite the Examiner to contact Applicants' attorney if the rejection is to be maintained since it is believed that, if necessary, an updated declaration can be provided.

Also, code segments are identified to correspond with the claim language of the general purpose register and processor and the sequencer coupled to the general purpose register and processor unit in operation of the sequencer and processor unit. For example, the code titled SQ CTL FO flow_seq.v corresponds to the sequencer control flow as understood by one of ordinary skill in the art. Given the nature of the design code and of the claimed invention, Applicants respectfully submit that the statement in combination with the contents and code in the exhibits themselves provide the detail as to the operation corresponding to the claims. Details are also provided in the declaration as to facts that existed and what facts occurred and when they occurred.

For example, the level of detail in the declaration identifies the portions of Exhibit B Chip Design Code that are believed to illustrate the reduction of practice of the claimed subject matter of these claims and is narrowed to reflect those portions believed to support Applicants' reduction to practice (Exhibit A also includes comments that explain the facts to support reduction to practice of the invention). In addition, there are statements within the code segments to assist a reader with the claim language. By way of example, in the SQ_export_alloc.v code segment, there are statements indicating for example, that the code "shows the SQ able to execute any types of export commands…an example of a shared resource is the instruction store, accesses to are controlled by: SQ_ctl_flow_seq.v identifying the code segment that performs the operation.

In an effort to advance prosecution, Applicants have attached a proposed replacement Exhibit A that illustrates claim language and corresponding code information with respect to claims including claims 2-5. Applicants respectfully request a review by the Examiner to determine whether the proposed Exhibit A provides adequate information which in combination with the second to the last paragraph of Applicants' affidavit would address the Examiner's concerns.

Claims 2 and 8 stand rejected under 35 U.S.C. § 112. Applicants respectfully traverse but have amended the claims to expedite prosecution. Claim 10 has also been amended to be consistent.

Claims 1-8, 10-11 and 15-16 stand rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by Lindholm. Applicants respectfully reassert the relevant remarks made in prior responses and also request reconsideration in view of Applicants' declaration, exhibits and remarks above.

Claims 1-8, 10-11 and 15-16 stand rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over U.S. patent No. 7,646,817 (Shen et al.) in view of U.S. Patent No. 6,697,074 (Parikh et al.). As to claim 1, the office action alleges that the use of the term "or" in Shen actually means "and" (office action page 19). However, Applicants respectfully note that the cited portion, namely col. 4, lns. 8-12 and FIG. 2 (reproduced below) contradicts this interpretation. The cited portion actually teaches a similar structure as shown in Applicants' prior art diagram (FIG. 3) showing a separate vertex shader and separate pixel shader. These are shown and described in Shen as being two separate shaders, a programmable vertex shader 212 and programmable pixel shader 214. The Shen reference is directed to accelerating video decoding using a graphics processing unit and when describing the GPU 208, it specifically

states that there are two different programmable shaders, a vertex shader, one "for performing graphics operations on a per-vertex basis, and a programmable pixel shader…for performing graphics operations on a per-pixel basis" (emphasis added). There is no teaching or suggestion of any unified shader. To the contrary, the Shen reference teaches a conventional independent vertex shader and pixel shader architecture. Since the reference does not teach what is alleged, Applicants respectfully submit that the claims are in condition for allowance. Other differences will be recognized by those of ordinary skill in the art.

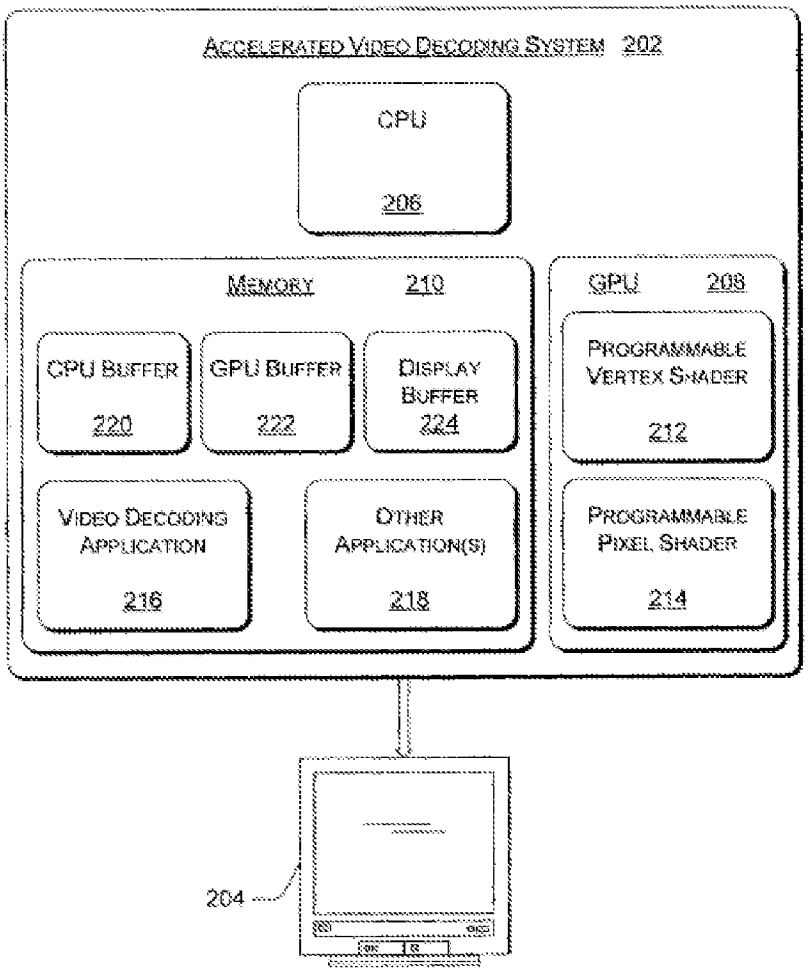


Figure 2

The dependent claims are believed to add additional novel and non-obvious subject matter.

As to claim 2, Applicants also respectfully note that the cited portion actually refers to "accelerated video decoding" and not to vertex calculations and pixel calculation operations facilitated by a sequencer that maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations. As noted above, the Shen reference and the cited portion thereof actually refers to the CPU executing video decoding application. There is no discussion of any sequencer that sequences instructions that cause the processor unit to execute vertex calculation and pixel calculation operations. Also, Applicants respectfully submit that the Parikh reference also fails to disclose a unified shader. The cited portion, namely col. 14, lns. 2-6 do not describe a unified shader comprising a general purpose register block for maintaining data. Also, there is no sequencer that is coupled to the general purpose register block wherein the sequencer maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block.

The cited portion of Parikh actually refers to the main processor 110 – not the graphics processor – that can start and stop the graphics and audio processors or determine a general status by reading status registers that are in the graphics and audio processors. What is described are standard status registers. In contrast, the claim recites a general purpose register block for maintaining data and a sequencer as claimed that maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block. There is no teaching or suggestion of this subject matter. Accordingly, Applicants respectfully request withdrawal of the rejection. Since

the references do not teach what they allege to teach and since the combination does not teach or suggest the subject matter, Applicants respectfully request withdrawal of the rejection.

Applicant respectfully submits that the claims are now believed to be in condition for allowance and that a timely Notice of Allowance be issued in this case. If the Examiner believes that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at (312) 356-5094.

Respectfully submitted,

Dated: June 6, 2013

By: /Christopher J. Reckamp/
Christopher J. Reckamp
Reg. No. 34,414

Faegre Baker Daniels LLP
311 S. Wacker Drive
Chicago, IL 60606
PHONE: (312) 356-5094
FAX: (312) 212-6501

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

A graphics processing architecture in one example performs vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continues pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

A graphics processing architecture in one example performs vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continues pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

1. A method comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor unless the general purpose register block does not have enough available space therein to store incoming vertex data; and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

Model of the “GPRs” (general purpose registers):

Reg_file.cpp

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//   Project: R400
//      File: reg_file.cpp
//
//   Description:
//      The GPRs
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/
#include "reg_file.h"


RegFile::RegFile()
{
       for (int i=0;i<128;i++)
              for (int j=0;j<16;j++)
                     for (int k=0;k<4;k++)
                            regValues[i].Val[j].field[k].clamp(0);
}

void RegFile::GetConstValues(const RegVect* &Values,int Addr)
{
       Values = &(regValues[Addr].Val[0]);
}

void RegFile::GetValues(RegVect* &Values,int Addr)
{
       Values = &(regValues[Addr].Val[0]);
}
```

Instruction store:

Instruction_store.cpp

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//   Project: R400
//      File: instruction_store.cpp
//
//   Description:
//      The instruction store
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//
#include "instruction_store.h"


IStore::IStore()
{
	for (int i=0;i<4096;i++)
	{
		instructions[i].byte0=0x00;
		instructions[i].byte1=0x00;
		instructions[i].byte2=0x00;
		instructions[i].byte3=0x00;
		instructions[i].byte4=0x00;
		instructions[i].byte5=0x00;
		instructions[i].byte6=0x00;
		instructions[i].byte7=0x00;
		instructions[i].byte8=0x00;
		instructions[i].byte9=0x00;
		instructions[i].byte10=0x00;
		instructions[i].byte11=0x00;
	}
}

void IStore::GetInst(Instruction &inst,int addr)
{
	inst = instructions[addr];
}

void IStore::GetInst(ALU_Instruction &aluInst, int addr)
{
	aluInst.SrcASel = ((instructions[addr].byte11 & 0x80) >> 7);
	aluInst.SrcBSel = ((instructions[addr].byte11 & 0x40) >> 6);
	aluInst.SrcCSel = ((instructions[addr].byte11 & 0x20) >> 5);
	aluInst.VectorOpcode = ((instructions[addr].byte11 & 0x1F));
	aluInst.SourceARegPointer = ((instructions[addr].byte10 ));
	aluInst.SourceBRegPointer = ((instructions[addr].byte9 ));
	aluInst.SourceCRegPointer = ((instructions[addr].byte8 ));
	aluInst.Constan0RelAbs = ((instructions[addr].byte7 & 0x80) >> 7);
	aluInst.Constan1RelAbs = ((instructions[addr].byte7 & 0x40) >> 6);
	aluInst.RelativeAddrRegSel = ((instructions[addr].byte7 & 0x20) >> 5);
	aluInst.PredicateSelect = ((instructions[addr].byte7 & 0x18) >> 3);
	aluInst.SourceANegate = ((instructions[addr].byte7 & 0x04) >> 2);
	aluInst.SourceBNegate = ((instructions[addr].byte7 & 0x02) >> 1);
	aluInst.SourceCNegate = ((instructions[addr].byte7 & 0x01)     );
	aluInst.SourceASwizzle = ((instructions[addr].byte6) );
	aluInst.SourceBSwizzle = ((instructions[addr].byte5) );
	aluInst.SourceCSwizzle = ((instructions[addr].byte4) );
	aluInst.ScalarOpcode = ((instructions[addr].byte3 & 0xfc) >> 2);
	aluInst.ScalarClamp = ((instructions[addr].byte3 & 0x02) >> 1);
	aluInst.VectorClamp = ((instructions[addr].byte3 & 0x01)     );
```

```
	aluInst.ScalarWriteMask = ((instructions[addr].byte2 & 0xf0) >> 4);
	aluInst.VectorWriteMask = ((instructions[addr].byte2 & 0x0f) );
	aluInst.ScalarResultPointer = ((instructions[addr].byte1 )     );
	aluInst.VectorResultPointer = ((instructions[addr].byte0 )     );
}

void IStore::GetInst(TInstrPacked &texInst, int addr)
{
	texInst.unpack((const uint8*)(&instructions[addr]));
}

void IStore::GetInst(CF_Instruction &cfInst, int addr, bool left)
{
	// read from bytes 11 thru 6
	if (left)
	{
		cfInst.opCode = ((instructions[addr].byte11 & 0xF0) >> 4);
		cfInst.addrMode = ((instructions[addr].byte11 & 0x08) >> 3);
		cfInst.bufferSel = ((instructions[addr].byte11 & 0x06) >> 1);
		cfInst.condition = ((instructions[addr].byte11 & 0x04) >> 2);
		cfInst.boolAddr = ((instructions[addr].byte11 & 0x03) << 6) |
		                  ((instructions[addr].byte10 & 0xFC) >> 2);
		cfInst.direction = ((instructions[addr].byte10 & 0x02) >> 1);
		cfInst.instTypeSer = ((instructions[addr].byte10 & 0x03) << 16) |
		                     ((instructions[addr].byte9) << 8) |
		                        ((instructions[addr].byte8));
		cfInst.predBreak = ((instructions[addr].byte8 & 0x20) >> 5);
		cfInst.loopId = ((instructions[addr].byte8 & 0x1F));
		cfInst.count = ((instructions[addr].byte7 & 0xF0) >> 4);
		cfInst.force = ((instructions[addr].byte7 & 0x20) >> 5);
		cfInst.jcAddress = ((instructions[addr].byte7 & 0x1F) << 8) |
		                   ((instructions[addr].byte6));
		cfInst.address = ((instructions[addr].byte7 & 0x0F) << 8) |
		                 ((instructions[addr].byte6));
		cfInst.allocSize = ((instructions[addr].byte6 & 0x0F));
	}
	// read from bytes 5 thru 0
	else
	{
		cfInst.opCode = ((instructions[addr].byte5 & 0xF0) >> 4);
		cfInst.addrMode = ((instructions[addr].byte5 & 0x08) >> 3);
		cfInst.bufferSel = ((instructions[addr].byte5 & 0x06) >> 1);
		cfInst.condition = ((instructions[addr].byte5 & 0x04) >> 2);
		cfInst.boolAddr = ((instructions[addr].byte5 & 0x03) << 6) |
		                  ((instructions[addr].byte4 & 0xFC) >> 2);
		cfInst.direction = ((instructions[addr].byte4 & 0x02) >> 1);
		cfInst.instTypeSer = ((instructions[addr].byte4 & 0x03) << 16) |
		                     ((instructions[addr].byte3) << 8) |
		                        ((instructions[addr].byte2));
		cfInst.predBreak = ((instructions[addr].byte2 & 0x20) >> 5);
		cfInst.loopId = ((instructions[addr].byte2 & 0x1F));
		cfInst.count = ((instructions[addr].byte1 & 0xF0) >> 4);
		cfInst.force = ((instructions[addr].byte1 & 0x20) >> 5);
		cfInst.jcAddress = ((instructions[addr].byte1 & 0x1F) << 8) |
		                   ((instructions[addr].byte0));
		cfInst.address = ((instructions[addr].byte1 & 0x0F) << 8) |
		                 ((instructions[addr].byte0));
		cfInst.allocSize = ((instructions[addr].byte0 & 0x0F));
```

```
	}
}

void IStore::SetInst(const Instruction &inst,int addr)
{
	instructions[addr]=inst;
}
```

Performing operations on pixels or vertices:

Arbiter.cpp

```
boolean Arbiter::chooseAluStation(int &lineNumber, Shader_Type &sType,
		bool otherAluRunning,const CfMachine& otherCFMachine,bool &predOn)
{
	int i;
	int vertexPick = -1;
	int pixelPick = -1;
	bool pcSpace;
	int lineCheck;
	predOn = true;

	// do pixels first
	lineCheck = pixelHead;
	for (i=0;i<pixelRsCount;i++)
	{
		if (pixelStation[lineCheck].status.valid != 0 &&
pixelStation[lineCheck].status.ressourceNeeded == ALU
		&& !pixelStation[lineCheck].status.event)
		{
			// no allocation needed
			if (pixelStation[lineCheck].status.allocation == SQ_NO_ALLOC)
			{
				pixelPick = lineCheck;
			}
			// we need to make sure there is space in the appropriate buffer
			else if (pixelStation[lineCheck].status.allocation == SQ_MEMORY &&
(pixelStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ->GetExportBuffer()/4
				&& pendingAllocs < 2 && sq->pSX_SQ->GetValid())
			{
				pixelPick = lineCheck;
			}
			else if (pixelStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL &&
				pixelStation[lineCheck].status.allocationSize <= sq->pSX_SQ-
>GetExportBuffer()/4
				&& pendingAllocs < 2 && sq->pSX_SQ->GetValid())
			{
				pixelPick = lineCheck;
			}
			// make sure the status says we can pick this pixel
			if (pixelPick != -1)
			{
				// check for serial with texture pending
				if (pixelStation[pixelPick].status.serial &&
					pixelStation[pixelPick].status.texReadsOutstanding)
					pixelPick = -1;
```

```
                        // if last or alloc is set we can only pick the two oldests
threads also for color exports
                        else if ((pixelStation[pixelPick].status.last
                                || pixelStation[pixelPick].status.allocation ==
SQ_PARAMETER_PIXEL )&&
                                !(pixelPick==pixelHead || pixelPick==((pixelHead-
1)%MAX_PIX_RESERVATION_SIZE)))
                                pixelPick = -1;
                        // cannot pick last if texture reads are outstanding
                        else if (pixelStation[pixelPick].status.last &&
                                pixelStation[pixelPick].status.texReadsOutstanding)
                                pixelPick = -1;
                        // can only pick the second to old if the first is already
running and last is set
                        else if (pixelStation[pixelPick].status.last && pixelHead !=
pixelPick)
                        {
                                if (pixelStation[pixelPick].status.first ||
!pixelStation[pixelHead].status.last
                                        || pixelStation[pixelHead].status.valid)
                                        pixelPick = -1;
                                else
                                {
                                        predOn = false;
                                        break;
                                }
                        }
                        else
                                break;
                }
        }// endif pixels

        lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
    }// end for loop

    lineCheck = vertexHead;
    for (i=0;i<vertexRsCount;i++)
    {
        if (vertexStation[lineCheck].status.valid != 0 &&
vertexStation[lineCheck].status.ressourceNeeded == ALU
                &&!vertexStation[lineCheck].status.event)
        {
                // no allocation needed
                if (vertexStation[lineCheck].status.allocation == SQ_NO_ALLOC)
                {
                        vertexPick = lineCheck;
                }
                // we need to make sure there is space in the appropriate buffer
                else
                {
                        if (vertexStation[lineCheck].status.allocation == SQ_MEMORY)
                        {
                                if
(((vertexStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ-
>GetExportBuffer()/4)
                                        && sq->pSX_SQ->GetValid() && pendingAllocs <2)
                                {
                                        vertexPick = lineCheck;
```

```
                                }
                        }
                        else if (vertexStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL)
                        {
                                // determine if there is space in the PCs for an
eventual PC export
                                pcSpace =
checkPC((vertexStation[lineCheck].status.allocationSize+1)*4);
                                if (pcSpace)
                                {
                                        // make sure every older threads have their
position allocated
                                        bool alloc_done = true;
                                        int alloc_line = vertexHead;
                                        while (lineCheck != alloc_line)
                                        {
                                                if
(vertexStation[alloc_line].status.pcAllocated == false)
                                                {
                                                        alloc_done = false;
                                                        break;
                                                }
                                                alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                        }
                                        if (alloc_done)
                                        {
                                                vertexPick = lineCheck;
                                        }
                                }
                        }
                        else if (vertexStation[lineCheck].status.allocation ==
SQ_POSITION
                                && (sq->pSX_SQ->GetPositionReady() >=
vertexStation[lineCheck].status.allocationSize )
                                && sq->pSX_SQ->GetValid()
                                && pendingAllocs <2)
                        {
                                // make sure every older threads have their position
allocated
                                bool alloc_done = true;
                                int alloc_line = vertexHead;
                                while (lineCheck != alloc_line)
                                {
                                        if
(vertexStation[alloc_line].status.posAllocated == false)
                                        {
                                                alloc_done = false;
                                                break;
                                        }
                                        alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                }
                                if (alloc_done)
                                {
                                        vertexPick = lineCheck;
                                }
```

```
                        }
                    }
                    // make sure the status says we can pick this vertex
                    if (vertexPick != -1)
                    {
                        // check for serial with texture pending
                        if (vertexStation[vertexPick].status.serial &&
                            vertexStation[vertexPick].status.texReadsOutstanding)
                            vertexPick = -1;
                        // if last is set we can only pick the two oldests threads
                        else if (vertexStation[vertexPick].status.last &&
                            !(vertexPick==vertexHead || vertexPick==((vertexHead-
1)%MAX_VTX_RESERVATION_SIZE)))
                            vertexPick = -1;
                        // cannot pick last if texture reads are outstanding
                        else if (vertexStation[vertexPick].status.last &&
                            vertexStation[vertexPick].status.texReadsOutstanding)
                            vertexPick = -1;
                        // can only pick the second to old if the first is already
running
                        else if ((vertexStation[vertexPick].status.last) && vertexHead
!= vertexPick)
                        {
                            if (vertexStation[vertexPick].status.first ||
!vertexStation[vertexHead].status.last
                                    || vertexStation[vertexHead].status.valid)
                                vertexPick = -1;
                            else
                            {
                                predOn = false;
                                break;
                            }
                        }
                        else
                            break;
                    }
                }// endif vertex

                lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
            }// end for loop

            // right now vertices have priority over pixels always,
            // will have to change this when the registers are there.
            if (vertexPick != -1)
            {
                lineNumber = vertexPick;
                sType = VERTEX;

                // HERE WE MUST DO THE ALLOCATION
                // also send a pulse to the SX if we need a buffer (position or multipass)

                if (vertexStation[vertexPick].status.allocation != SQ_NO_ALLOC)
                {
                    // parameter cache allocation
                    if (vertexStation[vertexPick].status.allocation ==
SQ_PARAMETER_PIXEL)
                    {
                        vertexStation[vertexPick].status.pcAllocated = true;
```

```
                        vertexStation[vertexPick].data.pcBasePtr = sq->pcHead;
                        vertexStation[vertexPick].data.exportId = 0;

                        if (sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4 < 128)
                        {
                                sq->pcHead = sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4;
                        }
                        else
                        {
                                sq->pcHead =
(vertexStation[vertexPick].status.allocationSize)*4-(128-sq->pcHead);
                                sq->checkHigh = !sq->checkHigh;
                        }
                        sq-
>pcAllocated.push((vertexStation[vertexPick].status.allocationSize)*4);
                }
                // position
                else if (vertexStation[vertexPick].status.allocation == SQ_POSITION)
                {
                        // starting a new allocation
                        pendingAllocs ++;

                        vertexStation[vertexPick].status.posAllocated = true;
                        vertexStation[vertexPick].status.pulseSx = true;
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = vertexStation[vertexPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        vertexStation[vertexPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 2;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = vertexStation[vertexPick].status.allocationSize-1;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }
                // multipass
                else
                {
                        // starting a new allocation
                        pendingAllocs ++;

                        vertexStation[vertexPick].status.pcAllocated = true;
                        vertexStation[vertexPick].status.pulseSx = true;
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = vertexStation[vertexPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        vertexStation[vertexPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 3;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
```

```
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = vertexStation[vertexPick].status.allocationSize;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }

                // dump the interface
                if (sq->m_dumpSQ > 0)
                {
                        sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
                        if (sq->m_sqSxDump->_data.Valid)
                        {
                                sq->m_sqSxDump->Dump();
                        }
                }

                // clear the allocation fields
                vertexStation[vertexPick].status.allocationSize = 0;
                vertexStation[vertexPick].status.allocation = SQ_NO_ALLOC;
        }
        return true;
}
if (pixelPick != -1)
{
        lineNumber = pixelPick;
        sType = PIXEL;

        if (pixelStation[pixelPick].status.allocation != SQ_NO_ALLOC)
        {
                // starting a new allocation
                pendingAllocs ++;

                if (pixelStation[pixelPick].status.allocation == SQ_PARAMETER_PIXEL)
                {
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = pixelStation[pixelPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        pixelStation[pixelPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;

                        sq->setContextNumber(st);
                        uint8 mode = sq->SQ_PROGRAM_CNTL.getPS_EXPORT_MODE();
                        // exporting Z
                        if (mode &0x01)
                                temp = 1;
                        // not exporting Z
                        else
                                temp = 0;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = pixelStation[pixelPick].status.allocationSize-temp-1;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }
                // multipass
                else
                {
```

```
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = pixelStation[pixelPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        pixelStation[pixelPick].data.exportId = exportId;
                        pixelStation[pixelPick].status.pulseSx = true;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 3;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = pixelStation[pixelPick].status.allocationSize;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                        pixelStation[pixelPick].status.pulseSx = true;
                }

                // dump the interface
                if (sq->m_dumpSQ > 0)
                {
                        sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
                        if (sq->m_sqSxDump->_data.Valid)
                        {
                                sq->m_sqSxDump->Dump();
                        }
                }

                // clear the allocation fields
                pixelStation[pixelPick].status.allocationSize = 0;
                pixelStation[pixelPick].status.allocation = SQ_NO_ALLOC;
        }
        return true;
    }

    return false;
}
```

Checking for GPR space:

Gpr_manager.cpp

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//   Project: R400
//      File: gpr_manager.cpp
//
//   Description:
//      The gpr manager.
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//
#include "gpr_manager.h"
#include "user_block_model.h"

GPR_manager::GPR_manager(cUSER_BLOCK_SQ *pSQ)
{
    // set the pointer to the SQ
```

```
	sq = pSQ;

	// set the limits (READ REGISTERS)
	pixLimit = sq->SQ_GPR_MANAGEMENT.REG_SIZE_PIX;
	vertLimit = 128-sq->SQ_GPR_MANAGEMENT.REG_SIZE_VTX;

	baseCountPix = 0;
	freeCountPix = 0;
	pixTestHigh = true;

	baseCountVert = 127;
	freeCountVert = 127;
	vertTestHigh = true;
}

boolean GPR_manager::testAllocate(int number_gpr,int &base_addr,Shader_Type stype)
{
	bool wrap = false;
	int testBaseCount;

	if (stype == PIXEL)
	{
		testBaseCount = baseCountPix;
		base_addr= baseCountPix;

		// special case for MAX GPRs
		if (number_gpr == pixLimit)
		{
			if (freeCountPix==baseCountPix && pixTestHigh &&
				freeCountPix != -1)
			{
				return false;
			}
			else
				return true;
		}

		if (testBaseCount + number_gpr < pixLimit)
			testBaseCount = testBaseCount + number_gpr;
		else
		{
			testBaseCount = number_gpr-(pixLimit-testBaseCount);
			// we wrapped change the test type
			pixTestHigh = !pixTestHigh;
			wrap = true;
		}
		if (pixTestHigh)
		{
			if (wrap)
				pixTestHigh = !pixTestHigh;
			if (testBaseCount >= freeCountPix && freeCountPix != -1)
			{
				// allocation succesfull
				return false;
			}
			else
			{
				// not enough space in GPRs
```

```
                        return true;
                }
        }
        else
        {
                if (wrap)
                        pixTestHigh = !pixTestHigh;
                if (testBaseCount <= freeCountPix && freeCountPix != -1)
                {
                        // allocation succesfull
                        return false;
                }
                else
                {
                        return true;
                }
        }
}
// vertices
else
{
        testBaseCount = baseCountVert;
        base_addr= baseCountVert;

        // special case for MAX GPRs
        if (number_gpr == -(vertLimit-128))
        {
                if (freeCountVert==baseCountVert && vertTestHigh &&
                        freeCountVert != -1)
                {
                        return false;
                }
                else
                        return true;
        }

        if (testBaseCount - number_gpr >= vertLimit)
                testBaseCount = testBaseCount - number_gpr;
        else
        {
                testBaseCount = 128-(number_gpr-(testBaseCount-vertLimit));
                // we wrapped change the test type
                vertTestHigh = !vertTestHigh;
                wrap = true;
        }
        if (vertTestHigh)
        {
                if (wrap)
                        vertTestHigh = !vertTestHigh;
                if (testBaseCount <= freeCountVert && freeCountVert != -1)
                {
                        // allocation succesfull
                        return false;
                }
                else
                {
                        return true;
                }
```

```
		}
		else
		{
			if (wrap)
				vertTestHigh = !vertTestHigh;
			if (testBaseCount >= freeCountVert && freeCountVert != -1)
			{
				// allocation succesfull
				return false;
			}
			else
			{
				return true;
			}
		}
	}
}

void GPR_manager::allocate(int number_gpr,int &base_addr,
                                                    Shader_Type stype)
{
	if (stype == PIXEL)
	{
		base_addr = baseCountPix;

		// special case for MAX GPRs
		if (number_gpr == pixLimit)
		{
			freeCountPix = -1;
		}

		if (baseCountPix + number_gpr < pixLimit)
			baseCountPix = base_addr + number_gpr;
		else
		{
			baseCountPix = number_gpr-(pixLimit-base_addr);
			// we wrapped change the test type
			pixTestHigh = !pixTestHigh;
		}
	}
	// vertices
	else
	{
		base_addr = baseCountVert;

		// special case for MAX GPRs
		if (number_gpr == -(vertLimit-128))
		{
			freeCountVert = -1;
		}

		if (baseCountVert - number_gpr >= vertLimit)
			baseCountVert = base_addr - number_gpr;
		else
		{
			baseCountVert = 128-(number_gpr-(base_addr-vertLimit));
			// we wrapped change the test type
			vertTestHigh = !vertTestHigh;
```

```
		}
	}
}

void GPR_manager::deAllocate(int number_gpr,Shader_Type stype)
{
	switch (stype)
	{
	case PIXEL:
		// special case for MAX GPRs
		if (number_gpr == pixLimit)
		{
			baseCountPix = 0;
			freeCountPix = 0;
			pixTestHigh = true;
			break;
		}
		if (freeCountPix + number_gpr < pixLimit)
			freeCountPix += number_gpr;
		else
		{
			freeCountPix = number_gpr-(pixLimit-freeCountPix);
			// we wrapped change the test type
			pixTestHigh = !pixTestHigh;
		}
		break;
	case VERTEX:
		// special case for MAX GPRs
		if (number_gpr == -(vertLimit-128))
		{
			baseCountVert = 127;
			freeCountVert = 127;
			vertTestHigh = true;
			break;
		}
		if (freeCountVert - number_gpr > vertLimit)
			freeCountVert -= number_gpr;
		else
		{
			freeCountVert = 128-(number_gpr-(freeCountVert-vertLimit));
			// we wrapped change the test type
			vertTestHigh = !vertTestHigh;
		}
		break;
	};
}
```

Write data to the GPRs:

Sq_block_model.cpp

```
// write to the SP dummy interface
RegVect* values;

regFile[j]->GetValues(values,address);

interpData.Address[i]=i+base_ptr;
```

```
interpData.NumParams = interp_params;

for (int k=0;k<16;k++)
{
	interpData.InterpData[i][k][j].field[0]=values[k].field[0];

	interpData.InterpData[i][k][j].field[1]=values[k].field[1];

	interpData.InterpData[i][k][j].field[2]=values[k].field[2];

	interpData.InterpData[i][k][j].field[3]=values[k].field[3];
}
// increment the GPR address
if (address+1 < gpr_manager->pixLimit)
{
	address ++;
}
else
{
	address = 0;
}
```

2. A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and

wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs and generates vertex position and appearance data in response to a selected one of the plurality of inputs.

GPR (general purpose register):

Reg_file.cpp

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//   Project: R400
//      File: reg_file.cpp
//
//   Description:
//      The GPRs
////%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
//
#include "reg_file.h"


RegFile::RegFile()
{
	for (int i=0;i<128;i++)
		for (int j=0;j<16;j++)
			for (int k=0;k<4;k++)
				regValues[i].Val[j].field[k].clamp(0);
}

void RegFile::GetConstValues(const RegVect* &Values,int Addr)
{
	Values = &(regValues[Addr].Val[0]);
}

void RegFile::GetValues(RegVect* &Values,int Addr)
{
	Values = &(regValues[Addr].Val[0]);
}
```

Processor unit:

The whole file sq_alu.cpp

Sequencer:

Arbiter.cpp (makes the decisions on who gets to run next)

Sq_block_model.cpp (connections to the SP)

Instruction_store.cpp (the instructions)

And the register block above mentioned.

Writing out color or position:

Sq_block_model.cpp

```
//****************************************************************************
// Output function for block
//****************************************************************************
void cUSER_BLOCK_SQ::Output(void)
{
	int i;
	static int current_export = 0;
	static int export_count = 0;
	static int currentPtr[4];

	if (outBuffer.valid)
	{
		outBuffer.valid = false;
		// VERTEX  PARAMETER CACHE EXPORT
```

```
		if ((outputType == VERTEX) && (currentExportDest < 16))
		{
			int pcPointer;
			// new export block reset the counts
			currentPtr[0] = currentAluPC;
			currentPtr[1] =
(currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1))%128;
			currentPtr[2] =
(currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1)*2)%128;
			currentPtr[3] =
(currentAluPC+(SQ_PROGRAM_CNTL.getVS_EXPORT_COUNT()+1)*3)%128;

			// set pcPointer to the correct value
			pcPointer = (currentPtr[current_export] + currentExportDest)%128;

			// copy data to the PCs
			int valid;
			for (i=0;i<16;i++)
			{
				valid = outBuffer.valids[i/4].getValue();
				if ((valid >> i%4) &0x01)
				{
					if (export_mask & 0x01)
						parameters[pcPointer].Val[i].field[0] =
outBuffer.values[i].field[0];
					if (export_mask & 0x02)
						parameters[pcPointer].Val[i].field[1] =
outBuffer.values[i].field[1];
					if (export_mask & 0x04)
						parameters[pcPointer].Val[i].field[2] =
outBuffer.values[i].field[2];
					if (export_mask & 0x08)
						parameters[pcPointer].Val[i].field[3] =
outBuffer.values[i].field[3];
				}
			}

			// dump the values to a file
			if(m_dumpSQ>0) {
				dumpPcValues(export_mask, pcPointer, outBuffer);
			}

			current_export++;
			if (current_export == 4)
			{
				current_export=0;
			}
		}// end parameter cache export
		// other exports
		else
		{
			pSP_SX->SetValid(true);
			for (i=0;i<16;i++)
			{
				pSP_SX->SetSP_SX_color(outBuffer.values[i].field[0],i*4);
				pSP_SX->SetSP_SX_color(outBuffer.values[i].field[1],i*4+1);
				pSP_SX->SetSP_SX_color(outBuffer.values[i].field[2],i*4+2);
				pSP_SX->SetSP_SX_color(outBuffer.values[i].field[3],i*4+3);
```

```
                        pSP_SX->SetSP_SX_exp_pvalid(outBuffer.valids[i/4],i/4);
                    }
                    uinteger<6> dest;
                    dest = currentExportDest;

                    pSP_SX->SetSP_SX_dest(dest);
                    pSP_SX->SetSP_SX_alu_id(currentExportAlu);
                    uinteger<2> exp_count;
                    exp_count = export_count;
                    pSP_SX->SetSP_SX_export_count(exp_count);
                    export_count = (export_count+1)%4;

                    pSP_SX->SetType(outputType);

                    if(m_dumpSQ>0) {
                       pSP_SX->GetNewAll(&(m_spSxDump->_data));
                       m_spSxDump->Dump();
                    }
                } // end other exports
        }
}
```

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

Processor unit

The whole file sq_alu.cpp

Shared resources are the instruction_store.cpp and register_file.cpp

Arbiter.cpp makes decisions who gets to run next be looking at the resources and needs:

```
boolean Arbiter::chooseAluStation(int &lineNumber, Shader_Type &sType,
                    bool otherAluRunning,const CfMachine& otherCFMachine,bool &predOn)
{
       int i;
       int vertexPick = -1;
       int pixelPick = -1;
       bool pcSpace;
       int lineCheck;
       predOn = true;

       // do pixels first
```

```
	lineCheck = pixelHead;
	for (i=0;i<pixelRsCount;i++)
	{
		if (pixelStation[lineCheck].status.valid != 0 &&
pixelStation[lineCheck].status.ressourceNeeded == ALU
			&& !pixelStation[lineCheck].status.event)
		{
			// no allocation needed
			if (pixelStation[lineCheck].status.allocation == SQ_NO_ALLOC)
			{
				pixelPick = lineCheck;
			}
			// we need to make sure there is space in the appropriate buffer
			else if (pixelStation[lineCheck].status.allocation == SQ_MEMORY &&
(pixelStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ->GetExportBuffer()/4
				&& pendingAllocs < 2 && sq->pSX_SQ->GetValid())
			{
				pixelPick = lineCheck;
			}
			else if (pixelStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL &&
				pixelStation[lineCheck].status.allocationSize <= sq->pSX_SQ-
>GetExportBuffer()/4
				&& pendingAllocs < 2 && sq->pSX_SQ->GetValid())
			{
				pixelPick = lineCheck;
			}
			// make sure the status says we can pick this pixel
			if (pixelPick != -1)
			{
				// check for serial with texture pending
				if (pixelStation[pixelPick].status.serial &&
					pixelStation[pixelPick].status.texReadsOutstanding)
					pixelPick = -1;
				// if last or alloc is set we can only pick the two oldests
threads also for color exports
				else if ((pixelStation[pixelPick].status.last
					|| pixelStation[pixelPick].status.allocation ==
SQ_PARAMETER_PIXEL )&&
					!(pixelPick==pixelHead || pixelPick==((pixelHead-
1)%MAX_PIX_RESERVATION_SIZE)))
					pixelPick = -1;
				// cannot pick last if texture reads are outstanding
				else if (pixelStation[pixelPick].status.last &&
					pixelStation[pixelPick].status.texReadsOutstanding)
					pixelPick = -1;
				// can only pick the second to old if the first is already
running and last is set
				else if (pixelStation[pixelPick].status.last && pixelHead !=
pixelPick)
				{
					if (pixelStation[pixelPick].status.first ||
!pixelStation[pixelHead].status.last
						|| pixelStation[pixelHead].status.valid)
						pixelPick = -1;
					else
					{
						predOn = false;
```

```
                                        break;
                                }
                        }
                        else
                                break;
                }
        }// endif pixels

        lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
    }// end for loop

    lineCheck = vertexHead;
    for (i=0;i<vertexRsCount;i++)
    {
        if (vertexStation[lineCheck].status.valid != 0 &&
vertexStation[lineCheck].status.ressourceNeeded == ALU
                &&!vertexStation[lineCheck].status.event)
        {
                // no allocation needed
                if (vertexStation[lineCheck].status.allocation == SQ_NO_ALLOC)
                {
                        vertexPick = lineCheck;
                }
                // we need to make sure there is space in the appropriate buffer
                else
                {
                        if (vertexStation[lineCheck].status.allocation == SQ_MEMORY)
                        {
                                if
(((vertexStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ-
>GetExportBuffer()/4)
                                        && sq->pSX_SQ->GetValid() && pendingAllocs <2)
                                {
                                        vertexPick = lineCheck;
                                }
                        }
                        else if (vertexStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL)
                        {
                                // determine if there is space in the PCs for an
eventual PC export
                                pcSpace =
checkPC((vertexStation[lineCheck].status.allocationSize+1)*4);
                                if (pcSpace)
                                {
                                        // make sure every older threads have their
position allocated
                                        bool alloc_done = true;
                                        int alloc_line = vertexHead;
                                        while (lineCheck != alloc_line)
                                        {
                                                if
(vertexStation[alloc_line].status.pcAllocated == false)
                                                {
                                                        alloc_done = false;
                                                        break;
                                                }
```

```
                                                alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                        }
                                        if (alloc_done)
                                        {
                                                vertexPick = lineCheck;
                                        }
                                }
                        }
                        else if (vertexStation[lineCheck].status.allocation ==
SQ_POSITION
                                && (sq->pSX_SQ->GetPositionReady() >=
vertexStation[lineCheck].status.allocationSize )
                                && sq->pSX_SQ->GetValid()
                                && pendingAllocs <2)
                        {
                                // make sure every older threads have their position
allocated
                                bool alloc_done = true;
                                int alloc_line = vertexHead;
                                while (lineCheck != alloc_line)
                                {
                                        if
(vertexStation[alloc_line].status.posAllocated == false)
                                        {
                                                alloc_done = false;
                                                break;
                                        }
                                        alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                }
                                if (alloc_done)
                                {
                                        vertexPick = lineCheck;
                                }
                        }
                }
                // make sure the status says we can pick this vertex
                if (vertexPick != -1)
                {
                        // check for serial with texture pending
                        if (vertexStation[vertexPick].status.serial &&
                                vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        // if last is set we can only pick the two oldests threads
                        else if (vertexStation[vertexPick].status.last &&
                                !(vertexPick==vertexHead || vertexPick==((vertexHead-
1)%MAX_VTX_RESERVATION_SIZE)))
                                vertexPick = -1;
                        // cannot pick last if texture reads are outstanding
                        else if (vertexStation[vertexPick].status.last &&
                                vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        // can only pick the second to old if the first is already
running
                        else if ((vertexStation[vertexPick].status.last) && vertexHead
!= vertexPick)
                        {
```

```
                                    if (vertexStation[vertexPick].status.first ||
!vertexStation[vertexHead].status.last
                                          || vertexStation[vertexHead].status.valid)
                                          vertexPick = -1;
                                    else
                                    {
                                          predOn = false;
                                          break;
                                    }
                              }
                              else
                                    break;
                        }
                  }// endif vertex

                  lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
            }// end for loop

            // right now vertices have priority over pixels always,
            // will have to change this when the registers are there.
            if (vertexPick != -1)
            {
                  lineNumber = vertexPick;
                  sType = VERTEX;

                  // HERE WE MUST DO THE ALLOCATION
                  // also send a pulse to the SX if we need a buffer (position or multipass)

                  if (vertexStation[vertexPick].status.allocation != SQ_NO_ALLOC)
                  {
                        // parameter cache allocation
                        if (vertexStation[vertexPick].status.allocation ==
SQ_PARAMETER_PIXEL)
                        {
                              vertexStation[vertexPick].status.pcAllocated = true;
                              vertexStation[vertexPick].data.pcBasePtr = sq->pcHead;
                              vertexStation[vertexPick].data.exportId = 0;

                              if (sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4 < 128)
                              {
                                    sq->pcHead = sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4;
                              }
                              else
                              {
                                    sq->pcHead =
(vertexStation[vertexPick].status.allocationSize)*4-(128-sq->pcHead);
                                    sq->checkHigh = !sq->checkHigh;
                              }
                              sq-
>pcAllocated.push((vertexStation[vertexPick].status.allocationSize)*4);
                        }
                        // position
                        else if (vertexStation[vertexPick].status.allocation == SQ_POSITION)
                        {
                              // starting a new allocation
                              pendingAllocs ++;
```

```
                        vertexStation[vertexPick].status.posAllocated = true;
                        vertexStation[vertexPick].status.pulseSx = true;
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = vertexStation[vertexPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        vertexStation[vertexPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 2;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = vertexStation[vertexPick].status.allocationSize-1;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }
                // multipass
                else
                {
                        // starting a new allocation
                        pendingAllocs ++;

                        vertexStation[vertexPick].status.pcAllocated = true;
                        vertexStation[vertexPick].status.pulseSx = true;
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = vertexStation[vertexPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        vertexStation[vertexPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 3;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = vertexStation[vertexPick].status.allocationSize;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }

                // dump the interface
                if (sq->m_dumpSQ > 0)
                {
                        sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
                        if (sq->m_sqSxDump->_data.Valid)
                        {
                                sq->m_sqSxDump->Dump();
                        }
                }

                // clear the allocation fields
                vertexStation[vertexPick].status.allocationSize = 0;
                vertexStation[vertexPick].status.allocation = SQ_NO_ALLOC;
        }
        return true;
}
if (pixelPick != -1)
{
```

```
lineNumber = pixelPick;
sType = PIXEL;

if (pixelStation[pixelPick].status.allocation != SQ_NO_ALLOC)
{
	// starting a new allocation
	pendingAllocs ++;

	if (pixelStation[pixelPick].status.allocation == SQ_PARAMETER_PIXEL)
	{
		sq->pSQ_SX->SetValid(true);
		uinteger<3> st;
		st = pixelStation[pixelPick].data.state;
		sq->pSQ_SX->SetSQ_SX_exp_state(st);
		sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
		pixelStation[pixelPick].data.exportId = exportId;
		exportId = !exportId;
		uinteger<2> temp;

		sq->setContextNumber(st);
		uint8 mode = sq->SQ_PROGRAM_CNTL.getPS_EXPORT_MODE();
		// exporting Z
		if (mode &0x01)
			temp = 1;
		// not exporting Z
		else
			temp = 0;
		sq->pSQ_SX->SetSQ_SX_exp_type(temp);
		sq->pSQ_SX->SetSQ_SX_exp_valid(true);
		temp = pixelStation[pixelPick].status.allocationSize-temp-1;
		sq->pSQ_SX->SetSQ_SX_exp_number(temp);
	}
	// multipass
	else
	{
		sq->pSQ_SX->SetValid(true);
		uinteger<3> st;
		st = pixelStation[pixelPick].data.state;
		sq->pSQ_SX->SetSQ_SX_exp_state(st);
		sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
		pixelStation[pixelPick].data.exportId = exportId;
		pixelStation[pixelPick].status.pulseSx = true;
		exportId = !exportId;
		uinteger<2> temp;
		temp = 3;
		sq->pSQ_SX->SetSQ_SX_exp_type(temp);
		sq->pSQ_SX->SetSQ_SX_exp_valid(true);
		temp = pixelStation[pixelPick].status.allocationSize;
		sq->pSQ_SX->SetSQ_SX_exp_number(temp);
		pixelStation[pixelPick].status.pulseSx = true;
	}

	// dump the interface
	if (sq->m_dumpSQ > 0)
	{
		sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
		if (sq->m_sqSxDump->_data.Valid)
		{
```

```
                                sq->m_sqSxDump->Dump();
                            }
                        }

                        // clear the allocation fields
                        pixelStation[pixelPick].status.allocationSize = 0;
                        pixelStation[pixelPick].status.allocation = SQ_NO_ALLOC;
                    }
                    return true;
                }

                return false;
            }
```

4. A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

Same as claim 3.

5. A unified shader comprising:

a processor unit;

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

Same as claim 3.

6. The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.

For structure not already set forth above, the texture fetcher allows fetch from memory. The arbiter.cpp file picks the programs that need to fetch data in this function:

```
boolean Arbiter::chooseTexStation(int &lineNumber,Shader_Type &sType)
{
	int i;
	int vertexPick = -1;
	int pixelPick = -1;
	int lineCheck;

	// do pixels first
	lineCheck = pixelHead;
	for (i=0;i<pixelRsCount;i++)
	{
		if (pixelStation[lineCheck].status.valid &&
pixelStation[lineCheck].status.ressourceNeeded == TEXTURE
			&& !pixelStation[lineCheck].status.event)
		{
			pixelPick=lineCheck;
		}
		// enforce restrictions based on the status
		if (pixelPick != -1)
		{
			// no texture ops while texture reads are outstanding
			if (pixelStation[pixelPick].status.texReadsOutstanding)
				pixelPick = -1;
			else
				break;
		}

		lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
	}

	lineCheck = vertexHead;
	for (i=0;i<vertexRsCount;i++)
	{
		if (vertexStation[lineCheck].status.valid &&
vertexStation[lineCheck].status.ressourceNeeded == TEXTURE
			&& !vertexStation[lineCheck].status.event)
		{
			vertexPick=lineCheck;
		}

		// enforce restrictions based on the status
		if (vertexPick != -1)
		{
			// no texture ops while texture reads are outstanding
			if (vertexStation[vertexPick].status.texReadsOutstanding)
				vertexPick = -1;
			else
				break;
		}

		lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
	}

	if (vertexPick != -1)
```

```
	{
		lineNumber = vertexPick;
		sType = VERTEX;
		return true;
	}
	if (pixelPick != -1)
	{
		lineNumber = pixelPick;
		sType = PIXEL;
		return true;
	}

	return false;
}
```

Then fills in a request in this function:

```
void  Arbiter::fillTextureInterface(int textureInstAddr,int texturePhase,boolean last)
{
	const RegVect* txAddr;
	TXAddr address;
	uinteger<7> registerAddress;
	uinteger<7> writeAddress;
	uint8 maxSize;
	int basePtr = textureCFMachine.stationData->data.gprBase;

	sq->pSQ_TP->SetValid(true);

	// Get the instruction
	TInstrPacked inst;

	// set the state to the current running ALU
	sq->setContextNumber(textureCFMachine.stationData->data.state);

	sq->instructionStore.GetInst(inst,textureInstAddr);
	switch (textureCFMachine.sType)
	{
	case PIXEL:
		maxSize = sq->gpr_manager->pixLimit;
		// compute the addresses (read address)
		if ((inst.getSRC_GPR() + basePtr) < maxSize)
			registerAddress = inst.getSRC_GPR() + basePtr;
		else
			registerAddress = inst.getSRC_GPR()-(maxSize-basePtr);
		// write address
		if ((inst.getDST_GPR() + basePtr) < maxSize)
			writeAddress = inst.getDST_GPR() + basePtr;
		else
			writeAddress = inst.getDST_GPR()-(maxSize-basePtr);
		break;
	case VERTEX:
		maxSize = sq->gpr_manager->vertLimit;
		// compute the addresses (read address)
		if (( basePtr - inst.getSRC_GPR()) >= maxSize)
			registerAddress =  basePtr - inst.getSRC_GPR();
		else
```

```
                registerAddress = 128-(inst.getSRC_GPR()-(basePtr-maxSize));
            // write address
            if (( basePtr - inst.getDST_GPR()) >= maxSize)
                writeAddress =  basePtr - inst.getDST_GPR();
            else
                writeAddress = 128-(inst.getDST_GPR()-(basePtr-maxSize));
            break;
      };
      sq->regFile[texturePhase]->GetConstValues(txAddr,registerAddress);
      int i;
      for(i=0;i<16;i++)
      {
       //Do the swizzle for the TP
       inst.doSrcSwizzle( txAddr[i].field[0], txAddr[i].field[1], txAddr[i].field[2],
txAddr[i].field[3],
                        address.x, address.y, address.z );
            sq->pSQ_TP->SetSP_TP_fetch_addr(address,i);
      }
      for (i=0;i<4;i++)
      {
            uinteger<4> valids;
            valids = textureCFMachine.stationData->data.valids[texturePhase][i];
            // modify the mask to turn on any pixels that are off if not fetch valid
only
            if (!inst.getFETCH_VALID_ONLY())
            {
                if (valids.getValue() != 0)
                    valids = 0x0F;
            }

            // now modify the mask based on the predicate vector
            if (inst.getPRED_SELECT())
            {
                bool pred = (inst.getPRED_CONDITION() == 1);
                if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4])
                {
                    // kill the pixel
                    valids = valids.getValue() & 0xE;

                }
                if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4+1])
                {
                    // kill the pixel
                    valids = valids.getValue() & 0xD;

                }
                if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4+2])
                {
                    // kill the pixel
                    valids = valids.getValue() & 0xB;

                }
                if (pred != textureCFMachine.stationData-
>data.predicates[texturePhase*16+i*4+3])
                {
```

```
                        // kill the pixel
                        valids = valids.getValue() & 0x7;

                }
            }
            sq->pSQ_TP->SetSQ_TP_pix_mask(valids,i);

            // send the LOD correction bits
            uinteger<9> LODCorrect;
            LODCorrect = textureCFMachine.stationData-
>data.LodCorrect[texturePhase][i];
            sq->pSQ_TP->SetSQ_TP_lod_correct(LODCorrect,i);
        }
        sq->pSQ_TP->SetSQ_TP_write_gpr_index(writeAddress);
        sq->pSQ_TP->SetSQ_TP_last(last);
        uinteger<6> line;
        line = textureCFMachine.lineNumber;
        sq->pSQ_TP->SetSQ_TP_thread_id(line);
        sq->pSQ_TP->SetSQ_TP_type(textureCFMachine.sType);
        TConstPacked tpConst;
        sq->textureStateStore[textureCFMachine.stationData-
>data.state].GetConstTState(tpConst,inst.getCONST_INDEX());
        sq->pSQ_TP->SetSQ_TP_const(tpConst);
        sq->pSQ_TP->SetSQ_TP_instr(inst);
        uinteger<3> ctxId;
        ctxId = textureCFMachine.stationData->data.state;
        sq->pSQ_TP->SetSQ_TP_ctx_id(ctxId);

        if(sq->m_dumpSQ>0) {
        sq->pSQ_TP->GetNewAll(&(sq->m_sqTpDump->_data));
        sq->m_sqTpDump->Dump();
    }
}
```

7. The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.

This is the chooseAluStation function of arbiter.cpp which picks the next block of work (the “selection circuit”)

```
boolean Arbiter::chooseAluStation(int &lineNumber, Shader_Type &sType,
                    bool otherAluRunning,const CfMachine& otherCFMachine,bool &predOn)
{
        int i;
        int vertexPick = -1;
        int pixelPick = -1;
        bool pcSpace;
        int lineCheck;
        predOn = true;

        // do pixels first
        lineCheck = pixelHead;
        for (i=0;i<pixelRsCount;i++)
        {
```

```
		if (pixelStation[lineCheck].status.valid != 0 &&
pixelStation[lineCheck].status.ressourceNeeded == ALU
			&& !pixelStation[lineCheck].status.event)
		{
			// no allocation needed
			if (pixelStation[lineCheck].status.allocation == SQ_NO_ALLOC)
			{
				pixelPick = lineCheck;
			}
			// we need to make sure there is space in the appropriate buffer
			else if (pixelStation[lineCheck].status.allocation == SQ_MEMORY &&
(pixelStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ->GetExportBuffer()/4
				&& pendingAllocs < 2 && sq->pSX_SQ->GetValid())
			{
				pixelPick = lineCheck;
			}
			else if (pixelStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL &&
				pixelStation[lineCheck].status.allocationSize <= sq->pSX_SQ-
>GetExportBuffer()/4
				&& pendingAllocs < 2 && sq->pSX_SQ->GetValid())
			{
				pixelPick = lineCheck;
			}
			// make sure the status says we can pick this pixel
			if (pixelPick != -1)
			{
				// check for serial with texture pending
				if (pixelStation[pixelPick].status.serial &&
					pixelStation[pixelPick].status.texReadsOutstanding)
					pixelPick = -1;
				// if last or alloc is set we can only pick the two oldests
threads also for color exports
				else if ((pixelStation[pixelPick].status.last
					|| pixelStation[pixelPick].status.allocation ==
SQ_PARAMETER_PIXEL )&&
					!(pixelPick==pixelHead || pixelPick==((pixelHead-
1)%MAX_PIX_RESERVATION_SIZE)))
					pixelPick = -1;
				// cannot pick last if texture reads are outstanding
				else if (pixelStation[pixelPick].status.last &&
					pixelStation[pixelPick].status.texReadsOutstanding)
					pixelPick = -1;
				// can only pick the second to old if the first is already
running and last is set
				else if (pixelStation[pixelPick].status.last && pixelHead !=
pixelPick)
				{
					if (pixelStation[pixelPick].status.first ||
!pixelStation[pixelHead].status.last
						|| pixelStation[pixelHead].status.valid)
						pixelPick = -1;
					else
					{
						predOn = false;
						break;
					}
				}
```

```
                        else
                                break;
                }
        }// endif pixels

        lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
    }// end for loop

    lineCheck = vertexHead;
    for (i=0;i<vertexRsCount;i++)
    {
        if (vertexStation[lineCheck].status.valid != 0 &&
vertexStation[lineCheck].status.ressourceNeeded == ALU
                &&!vertexStation[lineCheck].status.event)
        {
                // no allocation needed
                if (vertexStation[lineCheck].status.allocation == SQ_NO_ALLOC)
                {
                        vertexPick = lineCheck;
                }
                // we need to make sure there is space in the appropriate buffer
                else
                {
                        if (vertexStation[lineCheck].status.allocation == SQ_MEMORY)
                        {
                                if
(((vertexStation[lineCheck].status.allocationSize+1)*4 <= sq->pSX_SQ-
>GetExportBuffer()/4)
                                        && sq->pSX_SQ->GetValid() && pendingAllocs <2)
                                {
                                        vertexPick = lineCheck;
                                }
                        }
                        else if (vertexStation[lineCheck].status.allocation ==
SQ_PARAMETER_PIXEL)
                        {
                                // determine if there is space in the PCs for an
eventual PC export
                                pcSpace =
checkPC((vertexStation[lineCheck].status.allocationSize+1)*4);
                                if (pcSpace)
                                {
                                        // make sure every older threads have their
position allocated
                                        bool alloc_done = true;
                                        int alloc_line = vertexHead;
                                        while (lineCheck != alloc_line)
                                        {
                                                if
(vertexStation[alloc_line].status.pcAllocated == false)
                                                {
                                                        alloc_done = false;
                                                        break;
                                                }
                                                alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                        }
                                        if (alloc_done)
```

```
                                        {
                                                vertexPick = lineCheck;
                                        }
                                }
                        }
                        else if (vertexStation[lineCheck].status.allocation ==
SQ_POSITION
                                && (sq->pSX_SQ->GetPositionReady() >=
vertexStation[lineCheck].status.allocationSize )
                                && sq->pSX_SQ->GetValid()
                                && pendingAllocs <2)
                        {
                                // make sure every older threads have their position
allocated
                                bool alloc_done = true;
                                int alloc_line = vertexHead;
                                while (lineCheck != alloc_line)
                                {
                                        if
(vertexStation[alloc_line].status.posAllocated == false)
                                        {
                                                alloc_done = false;
                                                break;
                                        }
                                        alloc_line =
(alloc_line+1)%MAX_VTX_RESERVATION_SIZE;
                                }
                                if (alloc_done)
                                {
                                        vertexPick = lineCheck;
                                }
                        }
                }
                // make sure the status says we can pick this vertex
                if (vertexPick != -1)
                {
                        // check for serial with texture pending
                        if (vertexStation[vertexPick].status.serial &&
                                vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        // if last is set we can only pick the two oldests threads
                        else if (vertexStation[vertexPick].status.last &&
                                !(vertexPick==vertexHead || vertexPick==((vertexHead-
1)%MAX_VTX_RESERVATION_SIZE)))
                                vertexPick = -1;
                        // cannot pick last if texture reads are outstanding
                        else if (vertexStation[vertexPick].status.last &&
                                vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        // can only pick the second to old if the first is already
running
                        else if ((vertexStation[vertexPick].status.last) && vertexHead
!= vertexPick)
                        {
                                if (vertexStation[vertexPick].status.first ||
!vertexStation[vertexHead].status.last
                                        || vertexStation[vertexHead].status.valid)
                                        vertexPick = -1;
```

```
                                        else
                                        {
                                                predOn = false;
                                                break;
                                        }
                                }
                                else
                                        break;
                        }
                }// endif vertex

                lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
        }// end for loop

        // right now vertices have priority over pixels always,
        // will have to change this when the registers are there.
        if (vertexPick != -1)
        {
                lineNumber = vertexPick;
                sType = VERTEX;

                // HERE WE MUST DO THE ALLOCATION
                // also send a pulse to the SX if we need a buffer (position or multipass)

                if (vertexStation[vertexPick].status.allocation != SQ_NO_ALLOC)
                {
                        // parameter cache allocation
                        if (vertexStation[vertexPick].status.allocation ==
SQ_PARAMETER_PIXEL)
                        {
                                vertexStation[vertexPick].status.pcAllocated = true;
                                vertexStation[vertexPick].data.pcBasePtr = sq->pcHead;
                                vertexStation[vertexPick].data.exportId = 0;

                                if (sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4 < 128)
                                {
                                        sq->pcHead = sq-
>pcHead+(vertexStation[vertexPick].status.allocationSize)*4;
                                }
                                else
                                {
                                        sq->pcHead =
(vertexStation[vertexPick].status.allocationSize)*4-(128-sq->pcHead);
                                        sq->checkHigh = !sq->checkHigh;
                                }
                                sq-
>pcAllocated.push((vertexStation[vertexPick].status.allocationSize)*4);
                        }
                        // position
                        else if (vertexStation[vertexPick].status.allocation == SQ_POSITION)
                        {
                                // starting a new allocation
                                pendingAllocs ++;

                                vertexStation[vertexPick].status.posAllocated = true;
                                vertexStation[vertexPick].status.pulseSx = true;
                                sq->pSQ_SX->SetValid(true);
```

```
                        uinteger<3> st;
                        st = vertexStation[vertexPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        vertexStation[vertexPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 2;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = vertexStation[vertexPick].status.allocationSize-1;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }
                // multipass
                else
                {
                        // starting a new allocation
                        pendingAllocs ++;

                        vertexStation[vertexPick].status.pcAllocated = true;
                        vertexStation[vertexPick].status.pulseSx = true;
                        sq->pSQ_SX->SetValid(true);
                        uinteger<3> st;
                        st = vertexStation[vertexPick].data.state;
                        sq->pSQ_SX->SetSQ_SX_exp_state(st);
                        sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
                        vertexStation[vertexPick].data.exportId = exportId;
                        exportId = !exportId;
                        uinteger<2> temp;
                        temp = 3;
                        sq->pSQ_SX->SetSQ_SX_exp_type(temp);
                        sq->pSQ_SX->SetSQ_SX_exp_valid(true);
                        temp = vertexStation[vertexPick].status.allocationSize;
                        sq->pSQ_SX->SetSQ_SX_exp_number(temp);
                }

                // dump the interface
                if (sq->m_dumpSQ > 0)
                {
                        sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
                        if (sq->m_sqSxDump->_data.Valid)
                        {
                                sq->m_sqSxDump->Dump();
                        }
                }

                // clear the allocation fields
                vertexStation[vertexPick].status.allocationSize = 0;
                vertexStation[vertexPick].status.allocation = SQ_NO_ALLOC;
        }
        return true;
}
if (pixelPick != -1)
{
        lineNumber = pixelPick;
        sType = PIXEL;

        if (pixelStation[pixelPick].status.allocation != SQ_NO_ALLOC)
```

```
{
	// starting a new allocation
	pendingAllocs ++;

	if (pixelStation[pixelPick].status.allocation == SQ_PARAMETER_PIXEL)
	{
		sq->pSQ_SX->SetValid(true);
		uinteger<3> st;
		st = pixelStation[pixelPick].data.state;
		sq->pSQ_SX->SetSQ_SX_exp_state(st);
		sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
		pixelStation[pixelPick].data.exportId = exportId;
		exportId = !exportId;
		uinteger<2> temp;

		sq->setContextNumber(st);
		uint8 mode = sq->SQ_PROGRAM_CNTL.getPS_EXPORT_MODE();
		// exporting Z
		if (mode &0x01)
			temp = 1;
		// not exporting Z
		else
			temp = 0;
		sq->pSQ_SX->SetSQ_SX_exp_type(temp);
		sq->pSQ_SX->SetSQ_SX_exp_valid(true);
		temp = pixelStation[pixelPick].status.allocationSize-temp-1;
		sq->pSQ_SX->SetSQ_SX_exp_number(temp);
	}
	// multipass
	else
	{
		sq->pSQ_SX->SetValid(true);
		uinteger<3> st;
		st = pixelStation[pixelPick].data.state;
		sq->pSQ_SX->SetSQ_SX_exp_state(st);
		sq->pSQ_SX->SetSQ_SX_exp_alu_id(exportId);
		pixelStation[pixelPick].data.exportId = exportId;
		pixelStation[pixelPick].status.pulseSx = true;
		exportId = !exportId;
		uinteger<2> temp;
		temp = 3;
		sq->pSQ_SX->SetSQ_SX_exp_type(temp);
		sq->pSQ_SX->SetSQ_SX_exp_valid(true);
		temp = pixelStation[pixelPick].status.allocationSize;
		sq->pSQ_SX->SetSQ_SX_exp_number(temp);
		pixelStation[pixelPick].status.pulseSx = true;
	}

	// dump the interface
	if (sq->m_dumpSQ > 0)
	{
		sq->pSQ_SX->GetNewAll(&(sq->m_sqSxDump->_data));
		if (sq->m_sqSxDump->_data.Valid)
		{
			sq->m_sqSxDump->Dump();
		}
	}
```

```
                // clear the allocation fields
                pixelStation[pixelPick].status.allocationSize = 0;
                pixelStation[pixelPick].status.allocation = SQ_NO_ALLOC;
            }
            return true;
        }

        return false;
}
```

8. The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs.

The color export function (sq_block_model.cpp)

```
            // other exports
            else
            {
                pSP_SX->SetValid(true);
                for (i=0;i<16;i++)
                {
                    pSP_SX->SetSP_SX_color(outBuffer.values[i].field[0],i*4);
                    pSP_SX->SetSP_SX_color(outBuffer.values[i].field[1],i*4+1);
                    pSP_SX->SetSP_SX_color(outBuffer.values[i].field[2],i*4+2);
                    pSP_SX->SetSP_SX_color(outBuffer.values[i].field[3],i*4+3);
                    pSP_SX->SetSP_SX_exp_pvalid(outBuffer.valids[i/4],i/4);
                }
                uinteger<6> dest;
                dest = currentExportDest;

                pSP_SX->SetSP_SX_dest(dest);
                pSP_SX->SetSP_SX_alu_id(currentExportAlu);
                uinteger<2> exp_count;
                exp_count = export_count;
                pSP_SX->SetSP_SX_export_count(exp_count);
                export_count = (export_count+1)%4;

                pSP_SX->SetType(outputType);

                if(m_dumpSQ>0) {
                   pSP_SX->GetNewAll(&(m_spSxDump->_data));
                   m_spSxDump->Dump();
                }
            } // end other exports
```

9. The shader of claim 5, wherein the processor unit executes vertex calculations while the pixel calculations are still in progress.

Same function as claim 7, the chooseAluStation function will find another appropriate block of code (vertex or pixels) to run while some other are stuck.

10. The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.

Same as claim 9.

11. The shader of claim 7, wherein the control signal is provided by an arbiter.

Claim 7 is already done by an arbiter (arbiter.cpp).

12. A graphics processor comprising:

a unified shader comprising a processor unit that executes vertex calculations while the pixel calculations are still in progress.

The chooseAluStation function which will pick the right operations to execute next based on who is ready and what is available.

13. The graphics processor of claim 12 wherein the unified shader comprises a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to

cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

Same support as for claims 1, 2 and 3, a sequencer coupled to a processor unit and the sequencer maintains instructions to cause the processor to execute pix/vtx calculations.

14. The graphics processor of claim 12 comprising a vertex block operative to fetch vertex information from memory.

Both pixels and vertices can use the texture block, as in the chooseTexStation function

```
boolean Arbiter::chooseTexStation(int &lineNumber,Shader_Type &sType)
{
	int i;
	int vertexPick = -1;
	int pixelPick = -1;
	int lineCheck;

	// do pixels first
	lineCheck = pixelHead;
	for (i=0;i<pixelRsCount;i++)
	{
		if (pixelStation[lineCheck].status.valid &&
pixelStation[lineCheck].status.ressourceNeeded == TEXTURE
			&& !pixelStation[lineCheck].status.event)
		{
			pixelPick=lineCheck;
		}
		// enforce restrictions based on the status
		if (pixelPick != -1)
		{
			// no texture ops while texture reads are outstanding
			if (pixelStation[pixelPick].status.texReadsOutstanding)
				pixelPick = -1;
			else
				break;
		}

		lineCheck = (lineCheck+1)%MAX_PIX_RESERVATION_SIZE;
	}

	lineCheck = vertexHead;
	for (i=0;i<vertexRsCount;i++)
	{
		if (vertexStation[lineCheck].status.valid &&
vertexStation[lineCheck].status.ressourceNeeded == TEXTURE
			&& !vertexStation[lineCheck].status.event)
		{
			vertexPick=lineCheck;
		}
```

```
                // enforce restrictions based on the status
                if (vertexPick != -1)
                {
                        // no texture ops while texture reads are outstanding
                        if (vertexStation[vertexPick].status.texReadsOutstanding)
                                vertexPick = -1;
                        else
                                break;
                }

                lineCheck = (lineCheck+1)%MAX_VTX_RESERVATION_SIZE;
        }

        if (vertexPick != -1)
        {
                lineNumber = vertexPick;
                sType = VERTEX;
                return true;
        }
        if (pixelPick != -1)
        {
                lineNumber = pixelPick;
                sType = PIXEL;
                return true;
        }

        return false;
}
```

15. A unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.

The sq_alu.cpp file executes ALU instructions on either pixels or vertices. The ChooseAluStation function picks the next block of data to execute based on the workload.

16. The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

Same as claim 15, the chooseAluFunction will pick programs at various degrees of completion…

Electronic Patent Application Fee Transmittal

Application Number:	13109738
Filing Date:	17-May-2011
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Filer:	Christopher J. Reckamp/Christine Wright
Attorney Docket Number:	00100.36.0001

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Extension - 3 months with $0 paid	1253	1	1400	1400

TCL 1002

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
			Total in USD ($)	**1400**

Electronic Acknowledgement Receipt

EFS ID:	15970324
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp/Christine Wright
Filer Authorized By:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001
Receipt Date:	06-JUN-2013
Filing Date:	17-MAY-2011
Time Stamp:	15:59:22
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$1400
RAM confirmation Number	2786
Deposit Account	020390
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		360001_Response.pdf	142277 44fc5b4217512c0588512f8a7a1af6d483821cb4	yes	12

Multipart Description/PDF files in .zip description

Document Description	Start	End
Amendment/Req. Reconsideration-After Non-Final Reject	1	1
Abstract	2	2
Claims	3	6
Applicant Arguments/Remarks Made in an Amendment	7	12

Warnings:					
Information:					
2	Abstract	360001_Abstract1.pdf	56919 d7fdbca0a561a1a9829c06a13bd44b8954c61124	no	1
Warnings:					
Information:					
3	Abstract	360001_Abstract2.pdf	60956 c3d3e798fae0ef71ab657dcf73a1d11632d8229b	no	1
Warnings:					
Information:					
4	Miscellaneous Incoming Letter	360001_ExhibitA.pdf	431849 ae5a7785ea58e404c94f44267eaa2e827091caed	no	39
Warnings:					
Information:					
5	Fee Worksheet (SB06)	fee-info.pdf	30642 8e3a1a90fba44676d03210a0ab0b7319ce28daa8	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	722643		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (09-11)
Approved for use through 1/31/2014. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875	Application or Docket Number 13/109,738	Filing Date 05/17/2011	☐ To be Mailed

ENTITY: ☒ LARGE ☐ SMALL ☐ MICRO

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	RATE ($)	FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $310 ($155 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).			
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))				
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT **06/06/2013**

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	* 12	Minus	** 20	= 0	X $80 =	0
Independent (37 CFR 1.16(h))	* 6	Minus	***6	= 0	X $420 =	0
☐ Application Size Fee (37 CFR 1.16(s))						
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))						
					TOTAL ADD'L FEE	0

AMENDMENT

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =	
Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))						
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))						
					TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

SLIE
/VERONICA AUGBURN/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**
If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

29153 7590 08/08/2013

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

EXAMINER	
CHEN, FRANK S	

ART UNIT	PAPER NUMBER
2677	

NOTIFICATION DATE	DELIVERY MODE
08/08/2013	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

inteas@faegrebd.com
michelle.davis@faegrebd.com
cynthia.payson@faegrebd.com

Office Action Summary	Application No. 13/109,738	Applicant(s) MOREIN ET AL.	
	Examiner FRANK CHEN	Art Unit 2677	AIA (First Inventor to File) Status No

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☐ Responsive to communication(s) filed on *6 June 2013*.

☐ A declaration(s)/affidavit(s) under **37 CFR 1.130(b)** was/were filed on _____.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

5)☒ Claim(s) *1-8,10,11,15 and 16* is/are pending in the application.

5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6)☐ Claim(s) _____ is/are allowed.

7)☒ Claim(s) *1-8,10,11,15 and 16* is/are rejected.

8)☐ Claim(s) _____ is/are objected to.

9)☐ Claim(s) _____ are subject to restriction and/or election requirement.

* If any claims have been determined allowable, you may be eligible to benefit from the **Patent Prosecution Highway** program at a participating intellectual property office for the corresponding application. For more information, please see http://www.uspto.gov/patents/init_events/pph/index.jsp or send an inquiry to PPHfeedback@uspto.gov.

Application Papers

10)☐ The specification is objected to by the Examiner.

11)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

Certified copies:

a)☐ All b)☐ Some * c)☐ None of the:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

3) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

4) ☐ Other: _____.

DETAILED ACTION

Claim Status

1. Claims 1-8, 10-11, and 15-16 are currently pending in this application.

2. Claims 2, 8, and 10 have been currently amended.

Specification

3. The object of the Abstract has been withdrawn because of submission of properly amended Abstract on June 6, 2013.

Declaration filed under 37 CFR 1.131

4. The declaration filed 1/18/2012 under 37 CFR 1.131 and the Reference submitted on 6/6/2013 have been considered and are effective to overcome the prior art of reference Lindholm (U.S. Patent No. 7,038,685 B1) (hereinafter "Lindholm '685).

Claim Amendment

5. Rejection of claims 2 and 8 under 35 U.S.C. 112 has been withdrawn due to proper amendment.

Claim Rejections - 35 USC § 102

6. The following is a quotation of the appropriate paragraphs of pre-AIA 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

7. Additionally, Claims 1-8, 10-11, and 15-16 are further rejected under 35 U.S.C. 103 as being unpatentable over Shen et al (U.S. Patent No. 7,646817B1) in view of Parikh et al. (U.S. Patent No. 6,697,074 B2).

8. Regarding Claim 1 (Original), Shen discloses **A method comprising: performing vertex manipulation operations and pixel manipulation operations** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.) **and performing vertex operations on the vertex data by a processor and** (*Col. 4, lines 8-12* reciting "Exemplary GPU 208 includes a programmable vertex shader 212 for performing graphics operations on a per-vertex basis, and a programmable pixel shader 214 for performing graphics operations on a per-pixel basis." The programmable vertex shader performs graphics operations on vertex data sent to it, thus the programmable vertex shader 212 processes vertex data and the vertex shader 212 is included within the GPU 208, which corresponds to a processor.)

continuing pixel calculation operations that are to be or are currently being performed by the processor (*Col. 6, lines 16* reciting "At block 322, video decoding application 216 directs the pixel shader component 214 of GPU 208 to perform color space conversion processing on the reconstructed image. Color space conversion processing is performed pixel-by-pixel to convert an image from a color space in which it was created (e.g., YUV) to a color space supported by display device 204 (e.g., RGB)." The color space conversion corresponds to pixel calculation operations that are to be performed by the processor because pixel shader is acting on pixel calculations that occur after vertex calculation operations and is performed by the pixel shader component of the GPU 208.) **based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.** (*Col. 4, lines 30-32* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer." Accelerated video decoding which includes per-pixel operations is described in computer-executable instructions. These instructions which are in the form of computer-executable instructions are used for execution. The computer-readable memory medium corresponds to the instruction store that stores the computer-executable instructions.)

While Shen does not explicitly disclose **by transmitting vertex data to a general purpose register block,** and **unless the general purpose register block does not have enough available space therein to store incoming vertex data;** Parikh does disclose **by transmitting vertex data to a general purpose register**

block, (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are general purpose registers for storing at least pixel and vertex data and additional data formats.)

unless the general purpose register block does not have enough available space therein to store incoming vertex data; (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." The number of registers available in the graphics processor will be finite and they may all be filled with only pixel format (pixel data). Therefore, if all the registers are all filled with non-vertex data, the processor may not read and process vertex format (vertex data).)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further

teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

9. Regarding Claim 2 (Currently amended), Shen discloses **a processor unit;** (*Col. 4, lines 8-12* reciting "Exemplary GPU 208 includes a programmable vertex shader 212 for performing graphics operations on a per-vertex basis, and a programmable pixel shader 214 for performing graphics operations on a per-pixel basis." The GPU 208, which corresponds to a processor.)

a sequencer, coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be

implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The memory storage devices corresponds to the sequencer because it stores the computer-executable instructions, such as application modules, which are in a sequence. The application modules may be located in local computer storage media and such local storage medium is coupled to the processor since it is accessible by the processor.)

wherein the processor unit executes instructions that generate a pixel color in response to selected data from the general purpose register block and generates vertex position and appearance data in response to selected data from the general purpose register block. (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to receiving a select input to perform. The special effects processing such as inverse telecine and scalling or fading corresponds to generating pixel color. GPU inherently or implicitly must have general purpose register blocks to perform

calculations. One of ordinary skill would recognize such features in a GPU at and before time of the filing of this application.)

While Shen does not explicitly disclose **A unified shader, comprising: a general purpose register block for maintaining data;** and **general purpose register block and the** Parikh does disclose **A unified shader, comprising: a general purpose register block for maintaining data;** (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are general purpose registers that can store at least pixel and vertex data and additional formats of data.)

general purpose register block and the (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are general purpose registers that can store at least pixel and vertex data and additional formats of data.)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics

processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

10. Regarding Claim 3 (Original), Shen discloses **A unified shader comprising:**

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation

operations. Here the “or” can be interpreted to include the meaning of “and” since “or” includes the meaning of “and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.)

While Shen does not disclose **shared resources, operatively coupled to the processor unit;** Parikh does disclose **shared resources, operatively coupled to the processor unit;** (*Col. 14, lines 2-6* reciting “Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors.” Thus, pixel and vertex data may be both written to registers that are within the graphics processor. Therefore, the registers within the graphics processors are shared resources that may be used to store at least pixel formats, vertex formats, and additional data formats.)

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations. (*Col. 14, lines 2-6* reciting “Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors.” The registers available in the graphics processor will be finite and they may all be filled with pixel format (pixel data). Therefore, if there is no empty registers left and the registers are all filled with non-vertex data, the processor may not read and process vertex format (vertex data).)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

11. Regarding Claim 4 (Original), Shen discloses **A unified shader comprising: a processor unit operative to perform vertex calculation operations and pixel calculation operations; and** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-

interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.)

Parikh discloses **shared resources, operatively coupled to the processor unit;** (*Col. 14, lines 2-6* reciting "Main processor 110 can also load a number of graphics values (e.g., transformation matrices, pixel formats, vertex formats, etc. by writing to registers within the graphics and audio processors." The registers available in the graphics processor will be finite and they may all be filled with pixel format (pixel data). Therefore, if there is no empty registers left and the registers are all filled with non-vertex data, the processor may not read and process vertex format (vertex data).)

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations. (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image

sharpening or blurring." The GPU can perform vertex-based or pixel-based special effects processing which corresponds to performing vertex manipulation operation and pixel manipulation operations. Here the "or" can be interpreted to include the meaning of "and" since "or" includes the meaning of "and." Nothing in the specification of Shen indicates an exclusive "or" meaning. In fact the GPU of Shen is shown to perform operations on per-pixel and per-vertex.)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

12. Regarding Claim 5 (Original), Shen further discloses **A unified shader comprising: a processor unit;** (*Col. 4, lines 8-12* reciting "Exemplary GPU 208 includes a programmable vertex shader 212 for performing graphics operations on a per-vertex basis, and a programmable pixel shader 214 for performing graphics operations on a per-pixel basis." The GPU 208, which corresponds to a processor.)

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store. (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The memory storage device corresponds to the sequencer because it stores the computer-executable instructions, such as application modules, which are in a sequence. The application modules may be located in local computer

storage media and such local storage medium is coupled to the processor since it is accessible by the processor. A sequencer is implicitly coupled to the GPU since the decoding application directs the GPU to perform pixel/vertex calculations and the instructions must be sequenced in order for the proper results to be obtained.)

It would have been obvious for one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen because both are drawn to analogous art. Shen discloses accelerated decoding of video bitstreams using a graphics processing unit (GPU). The GPU generates vertex data which are processed for vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. Parikh is drawn to a unique vertex representation allowing the graphics pipeline to retain vertex state information and to mix indexed and direct vertex values and attributes. Parikh further teaches that it is possible to store vertex format and pixel format and other graphics information into registers of the graphics processor. Thus, the GPU of Shen may be modified with the registers taught in Parikh so that the generated vertex values may be stored in the GPU for later access. One of ordinary skill in the art would be motivated to combine the registers of Parikh with the GPU of Shen in order to store the vertex data generated in Shen to more efficiently construct polygons and other graphical objects. Therefore, it would be obvious to one of ordinary skill in the art to combine the teachings of Parikh with the teachings of Shen.

13. Regarding Claim 6 (Original), Shen further discloses **The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a**

memory. (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The communications network based on communication protocols corresponds to circuitry operative to fetch the instructions from the remote computer storage media.)

14. Regarding Claim 7 (Original), Shen further discloses **The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.** (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing

environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The communications networks also corresponds to selective circuit that provides information to the memory storage devices.)

15. Regarding Claim 8 (Currently amended), Shen further discloses **The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected data.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." Fading in or out corresponds to generating a pixel color in response to the GPU 208 receiving directions (plurality of inputs) from the video decoding application 216. The GPU 208 must inherently or implicitly have data stored in the general purpose register blocks in order to perform.)

16. Regarding Claim 10 (Currently amended), Shen further discloses **The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected data.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be

directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." Both scaling and inverse telecine corresponds to vertex position (scaling) and appearance data (reverse telecine).)

17. Regarding Claim 11 (Original), Shen further discloses **The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." De-interlacing, inverse telecine, and scaling all correspond to vertex position (scaling) and appearance data (reverse telecine).)

18. Regarding Claim 15 (Original), Shen discloses **A unified shader comprising: a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload.** (*Col. 6, lines 39-45* reciting "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring. The GPU corresponds to a processor unit flexibly

controlled, and the reconstructed image corresponds to the workload since the reconstructed image will have varying numbers of vertex and pixel data to process.)

19. Regarding Claim 16 (Original), Shen further discloses **The shader of claim 15 comprising an instruction store and wherein the processor unit performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.** (*Col. 4, lines 29-44* reciting "Accelerated video decoding may be described in the general context of computer-executable instructions, such as application modules, being executed by a computer. Generally, application modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Video decoding application 216 may be implemented using any number of programming techniques and may be implemented in local computing environments or in distributed computing environments where tasks are performed by remote processing devices that are linked through various communications networks based on any number of communication protocols. In such a distributed computing environment, application modules may be located in both local and remote computer storage media including memory storage devices." The local and remote computer storage media including memory storage devices corresponds to the instruction store. Computer executable instructions corresponds to the vertex and pixel manipulation operations which is completed at various degrees according to the structure of the application module (stored instructions) on the storage device (instruction store).)

Response to Arguments

20. Applicant's arguments filed on June 6, 2013 have been fully considered but they are not persuasive. Regarding Claim 1, applicants argue that Shen does not disclose a "unified shader." First, claim 1 makes no mention of a "unified shader." Second, claim 1 recites a method of **performing vertex manipulation operation and pixel manipulation operations**, both of which are performed by Shen as disclosed in rejection of claim 1. The GPU in Shen has both vertex shader and a pixel shader and such GPU architecture allows that GPU to perform both the vertex manipulation operation and the pixel manipulation operations. This is clearly stated in Shen, as shown in the rejection of claim 1. Claim 1 as recited is not allowable.

21. Regarding Claim 2, the applicants argue that the cited portion refers to "accelerated video decoding" and not to vertex calculations and pixel calculations. (pg. 11 of Remarks) Essentially, the applicants are arguing that accelerated video decoding does not involving any vertex calculation and pixel calculations performed by a processing unit. However, this is contrary to the evidence recited in Shen. Shen in *Abstract* recites "An accelerated video decoding system utilizes a graphics processing unit to perform motion compensation, image reconstruction, and color space conversion processes, while utilizing a central processing unit to perform other decoding processes." Moreover, Shen discloses that video decoding directly involves pixel and vertex calculations on a processor by reciting at *col. 6, lines 39-45*: "At block 326, video decoding application 216 may optionally be configured to direct GPU 208 to perform special effects processing on the reconstructed image. For example, GPU 208 may be

directed to perform vertex-based or pixel-based special effects processing such as de-interlacing, inverse telecine, scaling, fading in or out, and image sharpening or blurring." Therefore, Shen clearly shows that accelerated video decoding involves a processor performing vertex and/or pixel calculations. It should be noted here that the video decoding application 216 is what causes the execution of the pixel/vertex calculations on the GPU. The decoding application 216 comprises computer-executable instructions (a sequence of instructions) and decoding application 216 is shown to be configured to direct the GPU. (*col.* 6, *lines* 39-41) Therefore, the decoding application 216 is a sequence of instructions that directs the GPU to perform vertex/pixel calculations. The sequencer is inherently in a GPU because as the decoding application 216 directs the GPU, the instructions of the decoding application 216 will be fed into the GPU's instruction registers which will then be used to direct the GPU to perform the vertex/pixel calculations. Therefore, a sequencer is inherent or implicit in any GPU.

22. Applicants also argue that Shen does not disclose a general purpose register block, but Shen does disclose a GPU and this GPU inherently comprises general purpose registers. This is also well-known hardware architecture. Therefore, the memory contains decoding application 216 is coupled to the GPU with the general purpose registers. Furthermore, Parikh discloses that a main processor 110 can load a number of graphical values including vertex/pixel data into the registers within a graphics processor. (*col.* 14, *lines* 2-6) Therefore, Parikh supports the fact that a GPU can be modified to comprise general purpose registers.

23. Furthermore, Shen in view of Parikh discloses a unified shader in the form of the

GPU in Shen. While Shen does not explicitly recite a "unified shader," the GPU in Shen as modified by Parikh is implicitly a unified shader as it unifies both the vertex shader and pixel shader into a single GPU and also inherently possesses the registered blocks and sequencer. Consequently claims 1-8, 10-11, and 15-16 remain rejected.

24. Examiner believes that one method of overcoming the cited prior art is to amend Claim 2 to recite a "unified shader comprising a processor which comprises a single shader capable of performing both vertex and pixel calculations." While Shen discloses a single graphics processor capable of performing vertex and pixel calculations, neither Shen nor Parikh discloses a processor with a single shader capable of performing both vertex and pixel processing calculations. However, the Examiner further cites to Morgan III et al. (U.S. Patent No. 7,015,909 B1) which discloses at *col.* 8, *lines* 16-34:

> First consider each component individually. The control logic 310 generally controls the process of compiling and executing shaders, in this example according to method 400. The control logic 310 does not necessarily have sole control over the entire process. At various points, control may be shared or transferred to other components. In some embodiments, the control logic 310 may also detect and/or resolve conflicts at run time. It may also combine multiple shaders into a larger shader and then execute the larger shader (which shall be referred to as a composite shader) instead of the many constituent shaders. For example, if multiple shaders are to be applied to the same object, the control logic 310 might construct a single composite shader that has the same effect as the original multiple shaders. The fragment assembler 320 is responsible for assembling shaders to be executed from their constituent fragments. The run-time compiler 330 is responsible for compiling shaders at run time. The graphics engine 340 executes the compiled shaders. (emphasis added)

25. Shen in view of Morgan shows unequivocally states that it would have been obvious to one of ordinary skill in the art to modify the programmable pixel and vertex shaders in Shen into a single shader that can perform both pixel and vertex shading on

the same graphical object and achieve the same results that can be achieved by the pixel and vertex shaders in separation. It would have been obvious and beneficial to combine the pixel shader with the vertex shader into a single shading unit since a single shading unit will work faster.

Conclusion

26. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

CONTACT

Any inquiry concerning this communication or earlier communications from the examiner should be directed to FRANK CHEN whose telephone number is (571)270-7993. The examiner can normally be reached on 8 - 5, Monday - Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kee Tung can be reached on (571) 272-7794. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/FRANK CHEN/
Examiner, Art Unit 2677

/KEE M TUNG/

Supervisory Patent Examiner, Art Unit 2677

Notice of References Cited	Application/Control No. 13/109,738	Applicant(s)/Patent Under Reexamination MOREIN ET AL.	
	Examiner FRANK CHEN	Art Unit 2677	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-7,015,909 B1	03-2006	Morgan III et al.	345/426
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	**Art Unit**
	FRANK CHEN	2677

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	07/12/2011	03/11/2012	11/30/2012	07/30/2013					
	1	✓	✓	✓	✓					
	2	✓	✓	✓	✓					
	3	✓	✓	✓	✓					
	4	✓	✓	✓	✓					
	5	✓	✓	✓	✓					
	6	✓	✓	✓	✓					
	7	✓	✓	✓	✓					
	8	✓	✓	✓	✓					
	9	✓	✓	-	-					
	10	✓	✓	✓	✓					
	11	✓	✓	✓	✓					
	12	✓	✓	-	-					
	13	✓	✓	-	-					
	14	✓	✓	-	-					
	15	✓	✓	✓	✓					
	16	✓	✓	✓	✓					

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	FRANK CHEN	2677

CPC- SEARCHED		
Symbol	Date	Examiner

CPC COMBINATION SETS - SEARCHED		
Symbol	Date	Examiner

US CLASSIFICATION SEARCHED			
Class	Subclass	Date	Examiner
345	501	7/12/11	DW
above	updated	3/11/12	DW

SEARCH NOTES		
Search Notes	Date	Examiner
Searched EAST (all databases) see search history printout	7/12/11	DW
Also see search histories for apps 12/791,597 and 11/842,256	7/12/11	DW
conducted inventor name search	7/12/11	DW
updated search in EAST (all databases) see search history printout	3/11/12	DW
updated search in EAST (all databases) see search history printout	11/30/2012	FC
updated search in EAST (all databases) see attached search history printout	7/29/2013	FC
performed Google Patent search	7/29/2013	FC
performed Google Books search	7/30/2013	FC
performed Safari Books search	7/30/2013	FC

INTERFERENCE SEARCH			
US Class/ CPC Symbol	US Subclass / CPC Group	Date	Examiner

/FRANK CHEN/ Examiner.Art Unit 2677	

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	24	GPU WITH sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:16
L2	0	GPU NEAR sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
L3	1873	processing WITH unit WITH sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
L4	1098	processing ADJ unit WITH sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
L5	34	processing ADJ unit NEAR sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
L6	3	"7015909".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 21:09
S1	108	single WITH shader WITH pixel WITH vertex	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 14:43
S2	94	unified ADJ shader$2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:28
S3	94	unified ADJ shader$2	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:28
S4	131	pixel WITH vertex WITH combination WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:43
S5	400	combination WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:52
S6	32	combination ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:52
S7	868	single WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO;	OR	ON	2012/11/29 20:47

			JPO; DERWENT; IBM_TDB			
S8	145	single NEAR shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:47
S9	127	single ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S10	108	single WITH shader WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S11	10	single NEAR shader WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S12	145	single NEAR shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:50
S13	0	combin3 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:06
S14	4	integrated WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:20
S15	0	simultaneou$4 WITH pixel WITH vertex WITH sahder	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:24
S16	23	simultaneou$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:24
S17	42	concurrent WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:25
S18	2	coexist WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:26
S19	0	contemporaneous WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:28
S20	1	contemporary WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:29
S21	0	synchron$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:32

S22	9	combined WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:33
S23	0	cumulat$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S24	4	composite WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S25	8	together WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S26	44	incorporat$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:38
S27	0	integration WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:43
S28	0	consolida$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S29	8	cooperat$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S30	0	undivided WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S31	381	one WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S32	23	one WITH only WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S33	108	single WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:45
S34	94	unified ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 09:03
S35	6	Lindoln	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S36	5041	Lindholm	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/30 10:04

TCL 1002

			IBM_TDB			
S37	215	Lindholm AND shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S38	138	Lindholm AND programmable WITH graphics WITH processor	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S39	297	graphics WITH processor WITH vertex WITH pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:09
S40	294	processor WITH vertex WITH pixel WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:48
S41	131	graphics WITH processor WITH vertex WITH pixel WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:54
S42	104	graphics WITH processor WITH vertex WITH pixel WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:54
S43	105	graphics WITH processor WITH vertex WITH pixel WITH processing	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:00
S44	1	graphics WITH processor WITH vertex WITH pixel WITH processing WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:01
S45	81	graphics WITH processor WITH vertex WITH pixel WITH operations	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:29
S46	0	graphics WITH processor WITH vertex WITH manipulation WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S47	0	processor WITH vertex WITH manipulation WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S48	21	processor WITH vertex WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S49	2	graphics WITH processor WITH dual WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:27
S50	116	GPU WITH vertex WITH pixel WITH (operations OR manipulation OR calculation)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:28
S51	14	vertex WITH data WITH	US-PGPUB; USPAT;	OR	ON	2012/11/30

		general WITH purpose WITH register	USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB			15:10
S52	17	vertex WITH data WITH general WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:11
S53	18	general WITH purpose WITH register WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:12
S54	10	((general$1purpose WITH register) OR GPR) WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:15
S55	18	general WITH purpose WITH register WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:19
S56	25	general WITH purpose WITH register SAME vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:19
S57	94	vertex WITH data WITH store WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:56
S58	43	vertex NEAR data WITH store WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:59
S59	121	vertex NEAR data WITH stor$3 WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:22
S60	4	vertex NEAR data WITH transmit WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:43
S61	19	vertex NEAR data WITH transmit$5 WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:43
S62	194	vertex WITH pixel WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 21:28
S63	2	"7646817".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/12/01 14:23
S64	2	"6697074".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/12/01 14:23
S65	0	vertex SAME pixel SAME manipulation WITH "same" WITH unit	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:39

S66	6	vertex SAME pixel SAME manipulation WITH unit	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:40
S67	26	pixel WITH vertex WITH shader SAME manipulation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:41
S68	2	"6353439".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:53
S69	1583	shader NEAR10 vertex NEAR10 pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:19
S70	1491	shader NEAR5 vertex NEAR5 pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:19
S71	590	shader NEAR vertex NEAR pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:19
S72	804254	shader ADJ vertex ADSJ pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:20
S73	68	shader ADJ vertex ADJ pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:20
S74	1	multi$1purpose ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:21
S75	0	singl ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:21
S76	137	single ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:21
S77	21	(single ADJ shader) SAME vertex SAME pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:23
S78	209	(single WITH shader) SAME vertex SAME pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:25
S79	22	(single NEAR shader) SAME vertex SAME pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:25
S80	2	"6417858".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2013/07/29 16:30

TCL 1002

			IBM_TDB			
S81	247	vertex WITH pixel WITH shader WITH (single OR combin$6)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:09
S82	363	unifi$4 WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:39
S83	129	unifi$4 ADJ shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:39
S84	145	unifi$4 NEAR shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:39
S85	2	"6417858".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 11:14
S86	3	"7015909".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:12
S87	307	composite NEAR shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:29
S88	183	composite ADJ shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30
S89	245	composit$4 NEAR (shader OR shading)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30
S90	48	composite NEAR (shader OR shading)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30
S91	30	composite ADJ (shader OR shading)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30

EAST Search History (Interference)

<This search history is empty>

7/30/2013 9:10:43 PM
C:\Users\fchen\Documents\EAST\Workspaces\13109738_20110216077_Morein_et_al.wsp

PTO/SB/26 (08-11)
Approved for use through 07/31/2012. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TERMINAL DISCLAIMER TO OBVIATE A DOUBLE PATENTING REJECTION OVER A "PRIOR" PATENT	Docket Number (Optional) 00100.36.0001

In re Application of: Stephen Morein et al.

Application No.: 13/109,738

Filed: May 17, 2011

For: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

The owner*, ATI Technologies ULC, of 100 percent interest in the instant application hereby disclaims, except as provided below, the terminal part of the statutory term of any patent granted on the instant application which would extend beyond the expiration date of the full statutory term of **prior patent** No. 6,897,871 as the term of said **prior patent** is presently shortened by any terminal disclaimer. The owner hereby agrees that any patent so granted on the instant application shall be enforceable only for and during such period that it and the **prior patent** are commonly owned. This agreement runs with any patent granted on the instant application and is binding upon the grantee, its successors or assigns.

In making the above disclaimer, the owner does not disclaim the terminal part of the term of any patent granted on the instant application that would extend to the expiration date of the full statutory term of the **prior patent**, "as the term of said **prior patent** is presently shortened by any terminal disclaimer," in the event that said **prior patent** later:

expires for failure to pay a maintenance fee;
is held unenforceable;
is found invalid by a court of competent jurisdiction;
is statutorily disclaimed in whole or terminally disclaimed under 37 CFR 1.321;
has all claims canceled by a reexamination certificate;
is reissued; or
is in any manner terminated prior to the expiration of its full statutory term as presently shortened by any terminal disclaimer.

Check either box 1 or 2 below, if appropriate.

1. ☐ For submissions on behalf of a business/organization (e.g., corporation, partnership, university, government agency, etc.), the undersigned is empowered to act on behalf of the business/organization.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

2. ☑ The undersigned is an attorney or agent of record. Reg. No. 34,414

/Christopher J. Reckamp/ — Signature
January 31, 2014 — Date

Christopher J. Reckamp — Typed or printed name

312-356-5094 — Telephone Number

☑ Terminal disclaimer fee under 37 CFR 1.20(d) included.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

*Statement under 37 CFR 3.73(b) is required if terminal disclaimer is signed by the assignee (owner).
Form PTO/SB/96 may be used for making this certification. See MPEP § 324.

This collection of information is required by 37 CFR 1.321. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

PTO/AIA/22 (03-13)
Approved for use through 3/31/2013. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a)

Docket Number (Optional): 100100.360001-04-US

Application Number: 13/109,738

Filed: May 17, 2011

For: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

Art Unit: 2628

Examiner: Frank S. Chen

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above-identified application.

The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):

	Fee	Small Entity Fee	Micro Entity Fee	
☐ One month (37 CFR 1.17(a)(1))	$200	$100	$50	$
☐ Two months (37 CFR 1.17(a)(2))	$600	$300	$150	$
☑ Three months (37 CFR 1.17(a)(3))	$1,400	$700	$350	$ 1400.00
☐ Four months (37 CFR 1.17(a)(4))	$2,200	$1,100	$550	$
☐ Five months (37 CFR 1.17(a)(5))	$3,000	$1,500	$750	$

☐ Applicant asserts small entity status. See 37 CFR 1.27.

☐ Applicant certifies micro entity status. See 37 CFR 1.29.
Form PTO/SB/15A or B or equivalent must either be enclosed or have been submitted previously.

☐ A check in the amount of the fee is enclosed.

☐ Payment by credit card. Form PTO-2038 is attached.

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☑ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number 02-0390.

☐ Payment made via EFS-Web.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

I am the

☐ applicant.

☑ attorney or agent of record. Registration number 34,414.

☐ attorney or agent acting under 37 CFR 1.34. Registration number _____.

/christopher j. reckamp/
Signature

February 4, 2014
Date

Christopher J. Reckamp
Typed or printed name

(312) 356-5094
Telephone Number

NOTE: This form must be signed in accordance with 37 CFR 1.33. See 37 CFR 1.4 for signature requirements and certifications. Submit multiple forms if more than one signature is required, see below*.

☐ * Total of _____ forms are submitted.

This collection of information is required by 37 CFR 1.136(a). The information is required to obtain or retain a benefit by the public, which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 6 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Mail Stop PCT, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Electronic Patent Application Fee Transmittal

Application Number:	13109738
Filing Date:	17-May-2011
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Filer:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Extension - 3 months with $0 paid	1253	1	1400	1400

TCL 1002

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
Statutory or Terminal Disclaimer	1814	1	160	160
		Total in USD ($)		**1560**

Electronic Acknowledgement Receipt

EFS ID:	18113671
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp
Filer Authorized By:	
Attorney Docket Number:	00100.36.0001
Receipt Date:	04-FEB-2014
Filing Date:	17-MAY-2011
Time Stamp:	17:35:36
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$1560
RAM confirmation Number	4574
Deposit Account	020390
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

TCL 1002

1	Response After Final Action	100100-360001-04-AMENDAF2-4-14.pdf	91915 7c0d9c15d2f302f0435f5c3c7714467f27f63160	no	7
Warnings:					
Information:					
2	Terminal Disclaimer Filed	100100-360001-04-TERMDIS.pdf	30955 623cc4cca4586f955188b790647bd83329139fdd	no	1
Warnings:					
Information:					
3	Extension of Time	100100-360001-04-EXT2-4-14.pdf	152116 3b49887e666d60a88510748cb45607f63afee3e7	no	2
Warnings:					
Information:					
4	Fee Worksheet (SB06)	fee-info.pdf	32210 3e104acba5727303621e94adabef3e5823c61db4	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	307196		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/AIA/22 (03-13)
Approved for use through 3/31/2013. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a)

Docket Number (Optional): 100100.360001-04-US

Application Number: 13/109,738

Filed: May 17, 2011

For: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

Art Unit: 2628

Examiner: Frank S. Chen

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above-identified application.

The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):

	Fee	Small Entity Fee	Micro Entity Fee	
☐ One month (37 CFR 1.17(a)(1))	$200	$100	$50	$
☐ Two months (37 CFR 1.17(a)(2))	$600	$300	$150	$
☑ Three months (37 CFR 1.17(a)(3))	$1,400	$700	$350	$ 1400.00
☐ Four months (37 CFR 1.17(a)(4))	$2,200	$1,100	$550	$
☐ Five months (37 CFR 1.17(a)(5))	$3,000	$1,500	$750	$

☐ Applicant asserts small entity status. See 37 CFR 1.27.

☐ Applicant certifies micro entity status. See 37 CFR 1.29.
Form PTO/SB/15A or B or equivalent must either be enclosed or have been submitted previously.

☐ A check in the amount of the fee is enclosed.

☐ Payment by credit card. Form PTO-2038 is attached.

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☑ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number 02-0390.

☐ Payment made via EFS-Web.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

I am the

☐ applicant.

☑ attorney or agent of record. Registration number 34,414.

☐ attorney or agent acting under 37 CFR 1.34. Registration number _____.

/christopher j. reckamp/
Signature

February 4, 2014
Date

Christopher J. Reckamp
Typed or printed name

(312) 356-5094
Telephone Number

NOTE: This form must be signed in accordance with 37 CFR 1.33. See 37 CFR 1.4 for signature requirements and certifications. Submit multiple forms if more than one signature is required, see below*.

☐ * Total of _____ forms are submitted.

This collection of information is required by 37 CFR 1.136(a). The information is required to obtain or retain a benefit by the public, which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 6 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Mail Stop PCT, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Stephen Morein et al.	Examiner:	Frank S. Chen
Serial No.:	13/109,738	Art Unit:	2677
Filing Date:	May 17, 2011	Docket No.:	00100.36.0001

Confirmation No.: 2020

Title: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

AFTER FINAL RESPONSE

Dear Sir:

In response to the final office action dated August 8, 2013, and in view of discussions with the Examiner, Applicants petition for a three month extension of time and respond as follows:

Amendments to the Claims begins on page 2 of this paper.

Remarks begin on page 6 of this paper.

Amendments to the Claims:

Rewrite the claims as set forth below. This listing of claims replaces all prior versions and listings of claims in the application:

Listing of the Claims:

1. (currently amended) A method carried out by a unified shader comprising:

performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor within the unified shader unless the general purpose register block does not have enough available space therein to store incoming vertex data; and

continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.

2. (previously presented) A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and

wherein the processor unit executes instructions that generate a pixel color in response to selected data from the general purpose register block and generates vertex position and appearance data in response to selected data from the general purpose register block.

3. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

4. (original) A unified shader comprising:

a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

5. (original) A unified shader comprising:

a processor unit;

a sequencer coupled to the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store.

6. (original) The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.

7. (original) The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.

8. (previously presented) The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected data.

9. (canceled)

10. (previously presented) The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected data.

11. (original) The shader of claim 7, wherein the control signal is provided by an arbiter.

12. – 14. (canceled)

15. (currently amended) A unified shader comprising:

a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload; and

an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

16. (canceled)

REMARKS

Applicants' attorney wishes to thank the Examiner for the courtesies extended during the telephone conferences of January 29 and 30, 2014. Per the conferences the following agreement was reached:

(a) The Examiner indicated that claims 2-8, 10 and 11 would be allowed;

(b) The Examiner indicated that claim 1 would be allowed if inherent subject matter was added indicating that the method was carried out by a unified shader and that the processor is within the unified shader;

(c) The Examiner indicated that claim 16 would be allowable if written in independent form;

(d) The Examiner indicated that a response to final action would be the appropriate filing in view of the above; and

(e) The Examiner indicated that a terminal disclaimer would be required with respect to parent patent 6,897,871.

In view of the above agreement, Applicants have amended claim 1 to include what is believed to be inherent subject matter. Claim 15 has been amended to include the subject matter of claim 16 to expedite prosecution and Applicants submit a terminal disclaimer herewith.

Applicants respectfully submit that the claims are now believed to be in condition for allowance and that a timely Notice of Allowance be issued in this case. If the Examiner believes that any of the above is incorrect, the Examiner is invited to telephone the undersigned at (312) 356-5094.

Respectfully submitted,

Dated: February 4, 2014

By: /christopher j. reckamp/
Christopher J. Reckamp
Reg. No. 34,414

Faegre Baker Daniels LLP
311 S. Wacker Drive
Chicago, IL 60606
PHONE: (312) 356-5094
FAX: (312) 212-6501

PTO/SB/06 (09-11)
Approved for use through 1/31/2014. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875	Application or Docket Number 13/109,738	Filing Date 05/17/2011	☐ To be Mailed

ENTITY: ☒ LARGE ☐ SMALL ☐ MICRO

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	RATE ($)	FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $310 ($155 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).			
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))				
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT **02/04/2014**

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	* 11	Minus	** 20	= 0	X $80 =	0
Independent (37 CFR 1.16(h))	* 6	Minus	***6	= 0	X $420 =	0
☐ Application Size Fee (37 CFR 1.16(s))						
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))						
					TOTAL ADD'L FEE	0

AMENDMENT

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =	
Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))						
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))						
					TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

LIE
/JOY J. DOBBS/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**
If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

29153 7590 02/06/2014

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

EXAMINER	
CHEN, FRANK S	

ART UNIT	PAPER NUMBER
2611	

NOTIFICATION DATE	DELIVERY MODE
02/06/2014	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

inteas@faegrebd.com
michelle.davis@faegrebd.com
cynthia.payson@faegrebd.com

Applicant-Initiated Interview Summary	Application No.	Applicant(s)	
	13/109,738	MOREIN ET AL.	
	Examiner	Art Unit	
	FRANK CHEN	2611	

All participants (applicant, applicant's representative, PTO personnel):

(1) *FRANK CHEN*. (3)_____.

(2) *Christopher J. Reckamp*. (4)_____.

Date of Interview: *1/29/2014*.

Type: ☒ Telephonic ☐ Video Conference
☐ Personal [copy given to: ☐ applicant ☐ applicant's representative]

Exhibit shown or demonstration conducted: ☐ Yes ☐ No.
If Yes, brief description: _____.

Issues Discussed ☐101 ☐112 ☐102 ☒103 ☐Others
(For each of the checked box(es) above, please describe below the issue and detailed description of the discussion)

Claim(s) discussed: *2*.

Identification of prior art discussed: *Shen, Morgan III*.

Substance of Interview
(For each issue discussed, provide a detailed description and indicate if agreement was reached. Some topics may include: identification or clarification of a reference or a portion thereof, claim interpretation, proposed amendments, arguments of any applied references etc...)

Applicant's representative and Examiner discussed possible amendments to claim 2 such as tying the processor unit to be a part of the unified shader in the body of the claim because unified shader is only in the preamble of claim 2. The representative argues, in general, that the shader is hardware and the claim recites this shader hardware as a single processor that is capable of doing both pixel shading and vertex shading. Examiner has cited to Shen to describe a system that has a GPU that can perform both pixel and vertex shading. However, representative believes Shen is different from the Applicant's application because applicant's unified shader has a CPU within which perform the dual shading and Shen only discloses a GPU, not within a unified shader, that does pixel and vertex shading. Examiner, however, believes that a GPU can correspond to the shader while still being a processing unit or that the video decoder can be viewed upon as a unified shader. Applicant's representative believes Examiner's claim construction in such manner is faulty or too broad. No agreement is reached at the time of the interview but the Examiner will discuss this interpretation with his Supervisor.

Applicant recordation instructions: The formal written reply to the last Office action must include the substance of the interview. (See MPEP section 713.04). If a reply to the last Office action has already been filed, applicant is given a non-extendable period of the longer of one month or thirty days from this interview date, or the mailing date of this interview summary form, whichever is later, to file a statement of the substance of the interview

Examiner recordation instructions: Examiners must summarize the substance of any interview of record. A complete and proper recordation of the substance of an interview should include the items listed in MPEP 713.04 for complete and proper recordation including the identification of the general thrust of each argument or issue discussed, a general indication of any other pertinent matters discussed regarding patentability and the general results or outcome of the interview, to include an indication as to whether or not agreement was reached on the issues raised.

☐ Attachment

/FRANK CHEN/ Examiner, Art Unit 2611	/KEE M TUNG/ Supervisory Patent Examiner, Art Unit 2611

Summary of Record of Interview Requirements

Manual of Patent Examining Procedure (MPEP), Section 713.04, Substance of Interview Must be Made of Record
A complete written statement as to the substance of any face-to-face, video conference, or telephone interview with regard to an application must be made of record in the application whether or not an agreement with the examiner was reached at the interview.

Title 37 Code of Federal Regulations (CFR) § 1.133 Interviews
Paragraph (b)

In every instance where reconsideration is requested in view of an interview with an examiner, a complete written statement of the reasons presented at the interview as warranting favorable action must be filed by the applicant. An interview does not remove the necessity for reply to Office action as specified in §§ 1.111, 1.135. (35 U.S.C. 132)

37 CFR §1.2 Business to be transacted in writing.
All business with the Patent or Trademark Office should be transacted in writing. The personal attendance of applicants or their attorneys or agents at the Patent and Trademark Office is unnecessary. The action of the Patent and Trademark Office will be based exclusively on the written record in the Office. No attention will be paid to any alleged oral promise, stipulation, or understanding in relation to which there is disagreement or doubt.

The action of the Patent and Trademark Office cannot be based exclusively on the written record in the Office if that record is itself incomplete through the failure to record the substance of interviews.

It is the responsibility of the applicant or the attorney or agent to make the substance of an interview of record in the application file, unless the examiner indicates he or she will do so. It is the examiner's responsibility to see that such a record is made and to correct material inaccuracies which bear directly on the question of patentability.

Examiners must complete an Interview Summary Form for each interview held where a matter of substance has been discussed during the interview by checking the appropriate boxes and filling in the blanks. Discussions regarding only procedural matters, directed solely to restriction requirements for which interview recordation is otherwise provided for in Section 812.01 of the Manual of Patent Examining Procedure, or pointing out typographical errors or unreadable script in Office actions or the like, are excluded from the interview recordation procedures below. Where the substance of an interview is completely recorded in an Examiners Amendment, no separate Interview Summary Record is required.

The Interview Summary Form shall be given an appropriate Paper No., placed in the right hand portion of the file, and listed on the "Contents" section of the file wrapper. In a personal interview, a duplicate of the Form is given to the applicant (or attorney or agent) at the conclusion of the interview. In the case of a telephone or video-conference interview, the copy is mailed to the applicant's correspondence address either with or prior to the next official communication. If additional correspondence from the examiner is not likely before an allowance or if other circumstances dictate, the Form should be mailed promptly after the interview rather than with the next official communication.

The Form provides for recordation of the following information:
- Application Number (Series Code and Serial Number)
- Name of applicant
- Name of examiner
- Date of interview
- Type of interview (telephonic, video-conference, or personal)
- Name of participant(s) (applicant, attorney or agent, examiner, other PTO personnel, etc.)
- An indication whether or not an exhibit was shown or a demonstration conducted
- An identification of the specific prior art discussed
- An indication whether an agreement was reached and if so, a description of the general nature of the agreement (may be by attachment of a copy of amendments or claims agreed as being allowable). Note: Agreement as to allowability is tentative and does not restrict further action by the examiner to the contrary.
- The signature of the examiner who conducted the interview (if Form is not an attachment to a signed Office action)

It is desirable that the examiner orally remind the applicant of his or her obligation to record the substance of the interview of each case. It should be noted, however, that the Interview Summary Form will not normally be considered a complete and proper recordation of the interview unless it includes, or is supplemented by the applicant or the examiner to include, all of the applicable items required below concerning the substance of the interview.

A complete and proper recordation of the substance of any interview should include at least the following applicable items:
1) A brief description of the nature of any exhibit shown or any demonstration conducted,
2) an identification of the claims discussed,
3) an identification of the specific prior art discussed,
4) an identification of the principal proposed amendments of a substantive nature discussed, unless these are already described on the Interview Summary Form completed by the Examiner,
5) a brief identification of the general thrust of the principal arguments presented to the examiner,
   (The identification of arguments need not be lengthy or elaborate. A verbatim or highly detailed description of the arguments is not required. The identification of the arguments is sufficient if the general nature or thrust of the principal arguments made to the examiner can be understood in the context of the application file. Of course, the applicant may desire to emphasize and fully describe those arguments which he or she feels were or might be persuasive to the examiner.)
6) a general indication of any other pertinent matters discussed, and
7) if appropriate, the general results or outcome of the interview unless already described in the Interview Summary Form completed by the examiner.

Examiners are expected to carefully review the applicant's record of the substance of an interview. If the record is not complete and accurate, the examiner will give the applicant an extendable one month time period to correct the record.

Examiner to Check for Accuracy

If the claims are allowable for other reasons of record, the examiner should send a letter setting forth the examiner's version of the statement attributed to him or her. If the record is complete and accurate, the examiner should place the indication, "Interview Record OK" on the paper recording the substance of the interview along with the date and the examiner's initials.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

29153 7590 02/14/2014

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

EXAMINER
CHEN, FRANK S

ART UNIT	PAPER NUMBER
2611	

DATE MAILED: 02/14/2014

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

TITLE OF INVENTION: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

APPLN. TYPE	ENTITY STATUS	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	UNDISCOUNTED	$960	$0	$0	$960	05/14/2014

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the ENTITY STATUS shown above. If the ENTITY STATUS is shown as SMALL or MICRO, verify whether entitlement to that entity status still applies.

If the ENTITY STATUS is the same as shown above, pay the TOTAL FEE(S) DUE shown above.

If the ENTITY STATUS is changed from that shown above, on PART B - FEE(S) TRANSMITTAL, complete section number 5 titled "Change in Entity Status (from status indicated above)".

For purposes of this notice, small entity fees are 1/2 the amount of undiscounted fees, and micro entity fees are 1/2 the amount of small entity fees.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

29153 7590 02/14/2014
ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

TITLE OF INVENTION: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

APPLN. TYPE	ENTITY STATUS	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	UNDISCOUNTED	$960	$0	$0	$960	05/14/2014

EXAMINER	ART UNIT	CLASS-SUBCLASS
CHEN, FRANK S	2611	345-501000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
- ❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
- ❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) The names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) The name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.
1 _____
2 _____
3 _____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ❑ Individual ❑ Corporation or other private group entity ❑ Government

4a. The following fee(s) are submitted:
- ❑ Issue Fee
- ❑ Publication Fee (No small entity discount permitted)
- ❑ Advance Order - # of Copies _____

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
- ❑ A check is enclosed.
- ❑ Payment by credit card. Form PTO-2038 is attached.
- ❑ The Director is hereby authorized to charge the required fee(s), any deficiency, or credits any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
- ❑ Applicant certifying micro entity status. See 37 CFR 1.29 — NOTE: Absent a valid certification of Micro Entity Status (see forms PTO/SB/15A and 15B), issue fee payment in the micro entity amount will not be accepted at the risk of application abandonment.
- ❑ Applicant asserting small entity status. See 37 CFR 1.27 — NOTE: If the application was previously under micro entity status, checking this box will be taken to be a notification of loss of entitlement to micro entity status.
- ❑ Applicant changing to regular undiscounted fee status. — NOTE: Checking this box will be taken to be a notification of loss of entitlement to small or micro entity status, as applicable.

NOTE: This form must be signed in accordance with 37 CFR 1.31 and 1.33. See 37 CFR 1.4 for signature requirements and certifications.

Authorized Signature _____ Date _____

Typed or printed name _____ Registration No. _____

PTOL-85 Part B (10-13) Approved for use through 10/31/2013. OMB 0651-0033 U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

29153 7590 02/14/2014

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

EXAMINER	
CHEN, FRANK S	

ART UNIT	PAPER NUMBER
2611	

DATE MAILED: 02/14/2014

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 0 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 0 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

OMB Clearance and PRA Burden Statement for PTOL-85 Part B

The Paperwork Reduction Act (PRA) of 1995 requires Federal agencies to obtain Office of Management and Budget approval before requesting most types of information from the public. When OMB approves an agency request to collect information from the public, OMB (i) provides a valid OMB Control Number and expiration date for the agency to display on the instrument that will be used to collect the information and (ii) requires the agency to inform the public about the OMB Control Number's legal significance in accordance with 5 CFR 1320.5(b).

The information collected by PTOL-85 Part B is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450. Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Notice of Allowability	Application No. 13/109,738	Applicant(s) MOREIN ET AL.	
	Examiner FRANK CHEN	Art Unit 2611	AIA (First Inventor to File) Status No

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *02/04/2014*.
   ☐ A declaration(s)/affidavit(s) under **37 CFR 1.130(b)** was/were filed on_____.
2. ☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.
3. ☒ The allowed claim(s) is/are *1-8,10,11 and 15*. As a result of the allowed claim(s), you may be eligible to benefit from the **Patent Prosecution Highway** program at a participating intellectual property office for the corresponding application. For more information, please see http://www.uspto.gov/patents/init_events/pph/index.jsp or send an inquiry to PPHfeedback@uspto.gov .
4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

   Certified copies:

   a) ☐ All b) ☐ Some *c) ☐ None of the:
   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).
6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☐ Notice of References Cited (PTO-892)
2. ☐ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____
3. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
4. ☐ Interview Summary (PTO-413), Paper No./Mail Date _____ .
5. ☐ Examiner's Amendment/Comment
6. ☒ Examiner's Statement of Reasons for Allowance
7. ☐ Other _____.

/FRANK CHEN/
Examiner, Art Unit 2611

1. The present application is being examined under the pre-AIA first to invent provisions.

DETAILED ACTION

Claim Status

2. Claims 9, 12-14, and 16 have been cancelled.

3. Claims 1-8, 10-11, and 15 are presently pending.

Terminal Disclaimer

4. The terminal disclaimer filed on 02/04/2014 disclaiming the terminal portion of any patent granted on this application which would extend beyond the expiration date of November 20, 2023 has been reviewed and is accepted. The terminal disclaimer has been recorded.

Allowable Subject Matter

5. Claims 1-8, 10-11, and 15 are allowed. The following is an examiner's statement of reasons for allowance: Claims 1 and 15, as amended, recite **an instruction store** that is not disclosed in the prior arts. Claims 2 and 5 recite **a sequencer** which is also not disclosed in the prior arts because this sequencer is within the unified shader. Claims 3 and 4 recite **shared resources** which are not disclosed in the prior arts. For these reasons claims 1-8, 10-11, and 15 are allowed.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

CONTACT

Any inquiry concerning this communication or earlier communications from the examiner should be directed to FRANK CHEN whose telephone number is (571)270-7993. The examiner can normally be reached on 8 - 5, Monday - Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kee Tung can be reached on (571) 272-7794. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/FRANK CHEN/
Examiner, Art Unit 2611

/KEE M TUNG/

Supervisory Patent Examiner, Art Unit 2611

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 2020

SERIAL NUMBER	FILING or 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
13/109,738	05/17/2011 RULE	345	2611	00100.36.0001

APPLICANTS

INVENTORS

Stephen Morein, Cambridge, MA;
Laurent Lefebvre, Lachgnaie, CANADA;
Andy Gruber, Arlington, MA;
Andi Skende, Shrewsbury, MA;

**** CONTINUING DATA ***************************

This application is a CON of 12/791,597 06/01/2010 ABN
which is a CON of 11/842,256 08/21/2007 ABN
which is a CON of 11/117,863 04/29/2005 PAT 7327369
which is a CON of 10/718,318 11/20/2003 PAT 6897871

**** FOREIGN APPLICATIONS ***************************

**** IF REQUIRED, FOREIGN FILING LICENSE GRANTED ****

05/27/2011

Foreign Priority claimed	☐ Yes ☑ No	☐ Met after Allowance	STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS	INDEPENDENT CLAIMS
35 USC 119(a-d) conditions met	☐ Yes ☑ No					
Verified and Acknowledged	/DANIEL C WASHBURN/ Examiner's Signature	Initials	MA	5	~~16~~ 11	~~7~~ 6

ADDRESS

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606
UNITED STATES

TITLE

GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

FILING FEE RECEIVED		
1970	FEES: Authority has been given in Paper No._____ to charge/credit DEPOSIT ACCOUNT No._____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

OK TO ENTER: /F.C./

02/06/2014

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Stephen Morein et al.	Examiner:	Frank S. Chen
Serial No.:	13/109,738	Art Unit:	2677
Filing Date:	May 17, 2011	Docket No.:	00100.36.0001

Confirmation No.: 2020

Title: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**

AFTER FINAL RESPONSE

Dear Sir:

In response to the final office action dated August 8, 2013, and in view of discussions with the Examiner, Applicants petition for a three month extension of time and respond as follows:

Amendments to the Claims begins on page 2 of this paper.

Remarks begin on page 6 of this paper.

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	1	"13109738"	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:02
L2	76	graphics WITH processor WITH vertex WITH pixel WITH shader AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:05
L3	62	unified ADJ shader$2 AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:06
L4	19	(single NEAR shader) SAME vertex SAME pixel AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:06
L5	19	(single NEAR shader) SAME vertex SAME pixel AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:06
L6	29	vertex NEAR data WITH store WITH register AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:06
L7	90	vertex NEAR data WITH stor$3 WITH register AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:11
L8	7	together WITH pixel WITH vertex WITH shader AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:11
L9	7	together WITH pixel WITH vertex WITH shader AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR	US-PGPUB; USPAT; USOCR; FPRS; EPO;	OR	ON	2014/02/06 17:11

		G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	JPO; DERWENT; IBM_TDB			
L10	8	cooperat$5 WITH pixel WITH vertex WITH shader AND ((G06T15/005 OR G06T15/80 OR G06T1/20 OR G09G5/363 OR G06T1/60 OR G06F3/14).CPC.)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2014/02/06 17:11
S1	108	single WITH shader WITH pixel WITH vertex	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 14:43
S2	94	unified ADJ shader$2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:28
S3	94	unified ADJ shader$2	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:28
S4	131	pixel WITH vertex WITH combination WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:43
S5	400	combination WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:52
S6	32	combination ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 17:52
S7	868	single WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:47
S8	145	single NEAR shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:47
S9	127	single ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/29 20:48

			IBM_TDB			
S10	108	single WITH shader WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S11	10	single NEAR shader WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:48
S12	145	single NEAR shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 20:50
S13	0	combin3 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:06
S14	4	integrated WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:20
S15	0	simultaneou$4 WITH pixel WITH vertex WITH sahder	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:24
S16	23	simultaneou$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:24
S17	42	concurrent WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:25
S18	2	coexist WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:26
S19	0	contemporaneous WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/29 22:28

			IBM_TDB			
S20	1	contemporary WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:29
S21	0	synchron$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:32
S22	9	combined WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:33
S23	0	cumulat$4 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S24	4	composite WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S25	8	together WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:35
S26	44	incorporat$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:38
S27	0	integration WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:43
S28	0	consolida$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S29	8	cooperat$5 WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/29 22:44

			IBM_TDB			
S30	0	undivided WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S31	381	one WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S32	23	one WITH only WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:44
S33	108	single WITH pixel WITH vertex WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/29 22:45
S34	94	unified ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 09:03
S35	6	Lindoln	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S36	5041	Lindholm	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S37	215	Lindholm AND shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S38	138	Lindholm AND programmable WITH graphics WITH processor	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:04
S39	297	graphics WITH processor WITH vertex WITH pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/30 10:09

			IBM_TDB			
S40	294	processor WITH vertex WITH pixel WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:48
S41	131	graphics WITH processor WITH vertex WITH pixel WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:54
S42	104	graphics WITH processor WITH vertex WITH pixel WITH shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 10:54
S43	105	graphics WITH processor WITH vertex WITH pixel WITH processing	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:00
S44	1	graphics WITH processor WITH vertex WITH pixel WITH processing WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:01
S45	81	graphics WITH processor WITH vertex WITH pixel WITH operations	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 12:29
S46	0	graphics WITH processor WITH vertex WITH manipulation WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S47	0	processor WITH vertex WITH manipulation WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S48	21	processor WITH vertex WITH pixel WITH calculation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:23
S49	2	graphics WITH processor WITH dual WITH pixel WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/30 13:27

			IBM_TDB			
S50	116	GPU WITH vertex WITH pixel WITH (operations OR manipulation OR calculation)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 13:28
S51	14	vertex WITH data WITH general WITH purpose WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:10
S52	17	vertex WITH data WITH general WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:11
S53	18	general WITH purpose WITH register WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:12
S54	10	((general$1purpose WITH register) OR GPR) WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:15
S55	18	general WITH purpose WITH register WITH vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:19
S56	25	general WITH purpose WITH register SAME vertex	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:19
S57	94	vertex WITH data WITH store WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:56
S58	43	vertex NEAR data WITH store WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 15:59
S59	121	vertex NEAR data WITH stor$3 WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2012/11/30 16:22

			IBM_TDB			
S60	4	vertex NEAR data WITH transmit WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:43
S61	19	vertex NEAR data WITH transmit$5 WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 16:43
S62	194	vertex WITH pixel WITH register	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/11/30 21:28
S63	2	"7646817".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/12/01 14:23
S64	2	"6697074".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2012/12/01 14:23
S65	0	vertex SAME pixel SAME manipulation WITH "same" WITH unit	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:39
S66	6	vertex SAME pixel SAME manipulation WITH unit	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:40
S67	26	pixel WITH vertex WITH shader SAME manipulation	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:41
S68	2	"6353439".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 15:53
S69	1583	shader NEAR10 vertex NEAR10 pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2013/07/29 16:19

			IBM_TDB			
S70	1491	shader NEAR5 vertex NEAR5 pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:19
S71	590	shader NEAR vertex NEAR pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:19
S72	804254	shader ADJ vertex ADSJ pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:20
S73	68	shader ADJ vertex ADJ pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:20
S74	1	multi$1purpose ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:21
S75	0	singl ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:21
S76	137	single ADJ shader	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:21
S77	21	(single ADJ shader) SAME vertex SAME pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:23
S78	209	(single WITH shader) SAME vertex SAME pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:25
S79	22	(single NEAR shader) SAME vertex SAME pixel	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2013/07/29 16:25

			IBM_TDB			
S80	2	"6417858".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/29 16:30
S81	247	vertex WITH pixel WITH shader WITH (single OR combin$6)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:09
S82	363	unifi$4 WITH shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:39
S83	129	unifi$4 ADJ shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:39
S84	145	unifi$4 NEAR shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 10:39
S85	2	"6417858".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 11:14
S86	3	"7015909".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:12
S87	307	composite NEAR shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:29
S88	183	composite ADJ shad$4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30
S89	245	composit$4 NEAR (shader OR shading)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT;	OR	ON	2013/07/30 14:30

			IBM_TDB			
S90	48	composite NEAR (shader OR shading)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30
S91	30	composite ADJ (shader OR shading)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 14:30
S92	24	GPU WITH sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:16
S93	0	GPU NEAR sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
S94	1873	processing WITH unit WITH sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
S95	1098	processing ADJ unit WITH sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
S96	34	processing ADJ unit NEAR sequencer	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 20:17
S97	3	"7015909".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2013/07/30 21:09

EAST Search History (Interference)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L11	3	(unified WITH shader WITH sequencer).clm.	US-PGPUB; USPAT; UPAD	OR	ON	2014/02/06 17:30
L12	22384	(instruction WITH store WITH processor).clm.	US-PGPUB; USPAT; UPAD	OR	ON	2014/02/06 17:31

L13	0	(unified WITH shader WITH vertex WITH pixwl WITH instruction WITH store).clm.	US-PGPUB; USPAT; UPAD	OR	ON	2014/02/06 17:31
L14	4	(unified WITH shader WITH vertex WITH pixel WITH instruction WITH store).clm.	US-PGPUB; USPAT; UPAD	OR	ON	2014/02/06 17:31
L15	4	(unified WITH shader WITH vertex WITH instruction WITH store).clm.	US-PGPUB; USPAT; UPAD	OR	ON	2014/02/06 17:31

2/ 6/ 2014 5:32:50 PM
C:\ Users\ fchen\ Documents\ EAST\ Workspaces\ 13109738_20110216077_Morein_et_al.wsp

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	**Art Unit**
	FRANK CHEN	2611

CPC		
Symbol	**Type**	**Version**

CPC Combination Sets				
Symbol	**Type**	**Set**	**Ranking**	**Version**

/FRANK CHEN/ Examiner.Art Unit 2611	02/06/2014	**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	11	
/KEE M TUNG/ Supervisory Patent Examiner.Art Unit 2611	02/10/2014	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	**Art Unit**
	FRANK CHEN	2611

US ORIGINAL CLASSIFICATION		INTERNATIONAL CLASSIFICATION										
CLASS	SUBCLASS	CLAIMED					NON-CLAIMED					
345	501	G	0	6	F	15 / 00 (2006.01.01)						
CROSS REFERENCE(S)												
CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)											

/FRANK CHEN/ Examiner.Art Unit 2611	02/06/2014	**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	11	
/KEE M TUNG/ Supervisory Patent Examiner.Art Unit 2611	02/10/2014	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	**Art Unit**
	FRANK CHEN	2611

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original
1	1														
2	2														
3	3														
4	4														
5	5														
6	6														
7	7														
8	8														
9	10														
10	11														
11	15														

/FRANK CHEN/ Examiner.Art Unit 2611	02/06/2014	**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	11	
/KEE M TUNG/ Supervisory Patent Examiner.Art Unit 2611	02/10/2014	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	13109738	MOREIN ET AL.
	Examiner	Art Unit
	FRANK CHEN	2611

CPC- SEARCHED		
Symbol	Date	Examiner
G06T15/005	2/6/2014	FC
G06T15/80	2/6/2014	FC
G06T1/20	2/6/2014	FC
G09G5/363	2/6/2014	FC
G06T1/60	2/6/2014	FC
G06F3/14	2/6/2014	FC

CPC COMBINATION SETS - SEARCHED		
Symbol	Date	Examiner

US CLASSIFICATION SEARCHED			
Class	Subclass	Date	Examiner
345	501	7/12/11	DW
above	updated	3/11/12	DW

SEARCH NOTES		
Search Notes	Date	Examiner
Searched EAST (all databases) see search history printout	7/12/11	DW
Also see search histories for apps 12/791,597 and 11/842,256	7/12/11	DW
conducted inventor name search	7/12/11	DW
updated search in EAST (all databases) see search history printout	3/11/12	DW
updated search in EAST (all databases) see search history printout	11/30/2012	FC
updated search in EAST (all databases) see attached search history printout	7/29/2013	FC
performed Google Patent search	7/29/2013	FC
performed Google Books search	7/30/2013	FC
performed Safari Books search	7/30/2013	FC
performed EAST searches in all databases.	2/6/2014	FC

INTERFERENCE SEARCH

/FRANK CHEN/ Examiner, Art Unit 2611	

US Class/ CPC Symbol	US Subclass / CPC Group	Date	Examiner
See Search History		2/6/2014	FC

/FRANK CHEN/ Examiner, Art Unit 2611	

Application Number	Application/Control No.	Applicant(s)/Patent under Reexamination	
	13/109,738	MOREIN ET AL.	

Document Code - DISQ	Internal Document – DO NOT MAIL

TERMINAL DISCLAIMER	☒ APPROVED	☐ DISAPPROVED
Date Filed : 02-04-2014	This patent is subject to a Terminal Disclaimer	

Approved/Disapproved by:

Dorethea Lawrence

U.S. Patent and Trademark Office

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

29153 7590 02/14/2014
ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

Christine A. Wright	(Depositor's name)
/Christine A. Wright/	(Signature)
May 14, 2014	(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	05/17/2011	Stephen Morein	00100.36.0001	2020

TITLE OF INVENTION: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

APPLN. TYPE	ENTITY STATUS	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	UNDISCOUNTED	$960	$0	$0	$960	05/14/2014

EXAMINER	ART UNIT	CLASS-SUBCLASS
CHEN, FRANK S	2611	345-501000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) The names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) The name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.
1 Faegre Baker Daniels LLP
2
3

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: ATI Technologies ULC
(B) RESIDENCE: (CITY and STATE OR COUNTRY): Markham, Ontario, Canada

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☑ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
☑ Issue Fee
☐ Publication Fee (No small entity discount permitted)
☐ Advance Order - # of Copies .....

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
☐ A check is enclosed.
☐ Payment by credit card. Form PTO-2038 is attached.
☑ The Director is hereby authorized to charge the required fee(s), any deficiency, or credits any overpayment, to Deposit Account Number 020390 (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
☐ Applicant certifying micro entity status. See 37 CFR 1.29
NOTE: Absent a valid certification of Micro Entity Status (see forms PTO/SB/15A and 15B), issue fee payment in the micro entity amount will not be accepted at the risk of application abandonment.
☐ Applicant asserting small entity status. See 37 CFR 1.27
NOTE: If the application was previously under micro entity status, checking this box will be taken to be a notification of loss of entitlement to micro entity status.
☐ Applicant changing to regular undiscounted fee status.
NOTE: Checking this box will be taken to be a notification of loss of entitlement to small or micro entity status, as applicable.

NOTE: This form must be signed in accordance with 37 CFR 1.31 and 1.33. See 37 CFR 1.4 for signature requirements and certifications.

Authorized Signature /Christopher J. Reckamp/ Date May 14, 2014
Typed or printed name Christopher J. Reckamp Registration No. 34,414

PTOL-85 Part B (10-13) Approved for use through 10/31/2013. OMB 0651-0033 U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Electronic Patent Application Fee Transmittal

Application Number:	13109738
Filing Date:	17-May-2011
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Filer:	Christopher J. Reckamp/Christine Wright
Attorney Docket Number:	00100.36.0001

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Utility Appl Issue Fee	1501	1	960	960
Extension-of-Time:				

TCL 1002

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
			Total in USD ($)	**960**

Electronic Acknowledgement Receipt

EFS ID:	19026187
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Christopher J. Reckamp/Christine Wright
Filer Authorized By:	Christopher J. Reckamp
Attorney Docket Number:	00100.36.0001
Receipt Date:	14-MAY-2014
Filing Date:	17-MAY-2011
Time Stamp:	10:12:10
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$960
RAM confirmation Number	7322
Deposit Account	020390
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Issue Fee Payment (PTO-85B)	360001_IssueFee.pdf	1600728 419321fbb717a5396d7ba4aee0042c35131b2a42	no	1
Warnings:					
Information:					
2	Fee Worksheet (SB06)	fee-info.pdf	30518 0de49cc091bc6e255d9a11384af5e6bf5c111481	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	1631246		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	ISSUE DATE	PATENT NO.	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/109,738	06/24/2014	8760454	00100.36.0001	2020

29153 7590 06/04/2014

ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
CHICAGO, IL 60606

ISSUE NOTIFICATION

The projected patent number and issue date are specified above.

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment is 0 day(s). Any patent to issue from the above-identified application will include an indication of the adjustment on the front page.

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Application Assistance Unit (AAU) of the Office of Data Management (ODM) at (571)-272-4200.

APPLICANT(s) (Please see PAIR WEB site http://pair.uspto.gov for additional applicants):

Stephen Morein, Cambridge, MA;
Laurent Lefebvre, Lachgnaie, CANADA;
Andy Gruber, Arlington, MA;
Andi Skende, Shrewsbury, MA;

The United States represents the largest, most dynamic marketplace in the world and is an unparalleled location for business investment, innovation, and commercialization of new technologies. The USA offers tremendous resources and advantages for those who invest and manufacture goods here. Through SelectUSA, our nation works to encourage and facilitate business investment. To learn more about why the USA is the best country in the world to develop technology, manufacture products, and grow your business, visit SelectUSA.gov.

POWER OF ATTORNEY FROM ASSIGNEE

ATI Technologies ULC, a corporation of Canada, having a principal place of business at 1 Commerce Valley Drive East, Markham, Ontario, Canada L3T 7X6, is assignee of the entire right, title, and interest for the United States of America (as defined in 35 U.S.C. §100), by reason of an Assignment to the Assignee executed on August 21, 2003, of an invention known as **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER** (Attorney Docket No. 1972.3770004), that is disclosed and claimed in a patent application of the same title by the inventors Stephen MOREIN, Laurent LEFEBVRE, Andy GRUBER and Andi SKENDE (said application has a Filing Date or a 371(c) Date of May 17, 2011, at the U.S. Patent and Trademark Office, and having Application Number 13/109,738).

For the purpose of PAIR, the **Customer Number is 91505**.

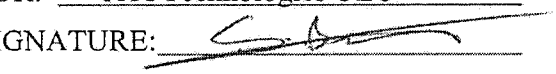
The Assignee hereby appoints the practitioners associated with **CUSTOMER NUMBER 91505** as my/our attorneys or agents to prosecute the application identified above, and any continuation, divisional, continuation-in-part, or reissue application thereof, and to transact all business in the U.S. Patent and Trademark Office connected therewith. The Assignee hereby grants said practitioners associated with **CUSTOMER NUMBER 91505** the power to insert on this Power of Attorney any further identification that may be necessary or desirable in order to comply with the rules of the U.S. Patent and Trademark Office.

Send correspondence to:

CUSTOMER NUMBER 91505

Direct phone calls to 202-371-2600.

FOR: ATI Technologies ULC

SIGNATURE: 

BY: Sorel Bosan

TITLE: Senior Patent Attorney

DATE: 6/ Oct/ 16

PTO/SB/96 (07-09)
Approved for use through 07/31/2012 OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

STATEMENT UNDER 37 CFR 3.73(b)

Atty. Docket No. 1972.3770004
ATI Reference No. 020001-US-CNT(4)

Applicant/Patent Owner: Stephen MOREIN et al.

Application No./Patent No.: 8,760,454 Filed/Issue Date: June 24, 2014

Titled: GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ATI Technologies ULC , a corporation
(Name of Assignee) (Type of Assignee, e.g., corporation, partnership, university, government agency, etc.

states that it is:

1. [X] the assignee of the entire right, title, and interest in;

2. [ ] an assignee of less than the entire right, title, and interest in
(The extent (by percentage) of its ownership interest is _____ %); or

3. [ ] the assignee of an undivided interest in the entirety of (a complete assignment from one of the joint inventors was made)

the patent application/patent identified above, by virtue of either:

A. [ ] An assignment from the inventor(s) of the patent application/patent identified above. The assignment was recorded in the United States Patent and Trademark Office at Reel _____, Frame _____, or for which a copy therefore is attached.

OR

B. [X] A chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as follows:

1. From: Inventors To: ATI Technologies, Inc.
The document was recorded in the United States Patent and Trademark Office at
Reel 032217, Frame 0137, or for which a copy thereof is attached.

2. From: ATI Technologies, Inc. To: ATI Technologies ULC
The document was recorded in the United States Patent and Trademark Office at
Reel 032265, Frame 0101, or for which a copy thereof is attached.

3. From: _____ To: _____
The document was recorded in the United States Patent and Trademark Office at
Reel _____, Frame _____, or for which a copy thereof is attached.

[ ] Additional documents in the chain of title are listed on a supplemental sheet(s).

[X] As required by 37 CFR 3.73(b)(1)(i), the documentary evidence of the chain of title from the original owner to the assignee was, or concurrently is being, submitted for recordation pursuant to 37 CFR 3.11.

[NOTE: A separate copy (*i.e.*, a true copy of the original assignment document(s)) must be submitted to Assignment Division in accordance with 37 CFR Part 3, to record the assignment in the records of the USPTO. See MPEP 302.08]

The undersigned (whose title is supplied below) is authorized to act on behalf of the assignee.

Signature — Date: 6/Oct/16

Sorel Bosan — Printed or Typed Name

Senior Patent Attorney — Title

This collection of information is required by 37 CFR 3.73(b). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Electronic Acknowledgement Receipt

EFS ID:	27158092
Application Number:	13109738
International Application Number:	
Confirmation Number:	2020
Title of Invention:	GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER
First Named Inventor/Applicant Name:	Stephen Morein
Customer Number:	29153
Filer:	Jonathan Tuminaro/Miyuki Christopher
Filer Authorized By:	Jonathan Tuminaro
Attorney Docket Number:	00100.36.0001
Receipt Date:	07-OCT-2016
Filing Date:	17-MAY-2011
Time Stamp:	16:57:23
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		19723770004_POA.pdf	436106 5cafbae1709740323dbcc926fb89173e2114 1085	yes	3

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Transmittal Letter	1	1
	Power of Attorney	2	2
	Assignee showing of ownership per 37 CFR 3.73	3	3
Warnings:			
Information:			
	Total Files Size (in bytes):	436106	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

JONATHAN TUMINARO
DIRECTOR
(202) 772-8967
JTUMINAR@SKGF.COM

Sterne Kessler Goldstein Fox
ATTORNEYS AT LAW

October 7, 2016

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Confirmation No. 2010
Art Unit 2611

Re: U.S. Patent No. 8,760,454; Issue Date: June 24, 2014
(from U.S. Appl. No. 13/109,738; Filing Date: May 17, 2011)
For: **GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER**
Inventors: MOREIN *et al.*
Our Ref: 1972.3770004

Commissioner:

Transmitted herewith for appropriate action are the following documents:

1. Power of Attorney; and

2. Statement Under 37 C.F.R § 3.73(b).

The above-listed documents are filed electronically.

The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency or credit any overpayment to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

Jonathan Tuminaro
Attorney for Patentees
Registration No. 61,327

JT/TJD/mic
Enclosures

4822171_1

TCL 1002

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
13/109,738	05/17/2011	Stephen Morein	1972.3770004

CONFIRMATION NO. 2020

POA ACCEPTANCE LETTER

91505
Sterne, Kessler, Goldstein & Fox P.L.L.C.
1100 New York Avenue N.W.
Washington, DC 20005

OC000000086491652

Date Mailed: 10/14/2016

NOTICE OF ACCEPTANCE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 10/07/2016.

The Power of Attorney in this application is accepted. Correspondence in this application will be mailed to the above address as provided by 37 CFR 1.33.

Questions about the contents of this notice and the requirements it sets forth should be directed to the Office of Data Management, Application Assistance Unit, at **(571) 272-4000** or **(571) 272-4200** or **1-888-786-0101**.

/qtran/

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
13/109,738	05/17/2011	Stephen Morein	00100.36.0001

CONFIRMATION NO. 2020

POWER OF ATTORNEY NOTICE

29153
ADVANCED MICRO DEVICES, INC.
C/O Faegre Baker Daniels LLP
311 S. WACKER DRIVE
Suite 4300
CHICAGO, IL 60606

OC00000086491572

Date Mailed: 10/14/2016

NOTICE REGARDING CHANGE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 10/07/2016.

- The Power of Attorney to you in this application has been revoked by the assignee who has intervened as provided by 37 CFR 3.71. Future correspondence will be mailed to the new address of record(37 CFR 1.33).

Questions about the contents of this notice and the requirements it sets forth should be directed to the Office of Data Management, Application Assistance Unit, at **(571) 272-4000** or **(571) 272-4200** or **1-888-786-0101**.

/qtran/

AO 120 (Rev. 08/10)

TO:	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK
Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Delaware on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
	1/23/2017	Delaware

PLAINTIFF	DEFENDANT
Advanced Micro Devices, Inc. and ATI Technologies ULC	LG Electronics, Inc., LG Electronics U.S.A., Inc., LG Electronics MobileComm U.S.A., Inc.

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	US 7,633,506	12/15/2009	ATI Technologies ULC
2	US 7,796,133	9/14/2010	ATI Technologies ULC
3	US 8,760,454	6/24/2014	ATI Technologies ULC
4			
5			

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director Copy 3—Upon termination of action, mail this copy to Director
Copy 2—Upon filing document adding patent(s), mail this copy to Director Copy 4—Case file copy

Trial@uspto.gov
571-272-7822

IPR2017-01225 Paper No. 10
IPR2017-01226 Paper No. 10
Date Entered: September 29, 2017

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LG ELECTRONICS, INC.,
Petitioner,

v.

ATI TECHNOLOGIES ULC,
Patent Owner.

Case IPR2017-01225
Patent 8,760,454 B2
Case IPR2017-01226
Patent 9,582,846 B2[1]

Before JONI Y. CHANG, BRIAN J. MCNAMARA, and JAMES B. ARPIN, *Administrative Patent Judges*.

McNAMARA, *Administrative Patent Judge*.

JUDGMENT
TERMINATION OF THE PROCEEDING

37 C.F.R. § 42.73

---

[1] This order is to be filed in each case. The parties are not authorized to use this style heading in any subsequent papers.

The parties have requested that these proceedings be terminated pursuant to a settlement. On September 28, 2017, the Board authorized the parties to file a Joint Motion To Terminate and a Joint Request that the Settlement Agreement Be Treated As Business Confidential Information under 37 C.F.R. 42.74(c). Paper 7 in IPR2017-01225; Paper 8 in IPR2017-01226. On September 28, 2017 the parties filed a Joint Motion Terminate each proceeding (Paper 8 in IPR2017-01225; Paper 9 in IPR2017-01226) and a Joint Request To File Settlement Agreement As Business Confidential in each proceeding (Paper 9 in IPR2017-01225; Paper 10 in IPR2017-01226). *See* 35 U.S.C. § 317(a); 37 C.F.R. § 42.72. The parties also filed a copy of a written settlement agreement. Ex. 1012 in IPR2017-01225; Ex. 1010 in IPR2017-01226.

A decision by the Board to institute a trial has not yet been entered. The parties have not identified any other related matters. Under these circumstances, the Board determines that it is appropriate to enter judgment[2] and terminate the proceedings without rendering a final written decision. 37 C.F.R. § 42.72.

It is, therefore,

ORDERED that the joint motions to terminate the proceedings are GRANTED and the proceedings are hereby terminated;

FURTHER ORDERED that the parties' joint requests that the settlement agreements be treated as business confidential information which shall be kept separate from the file of the involved patents under the provisions of 35 U.S.C. § 317(b) and 37 C.F.R. § 42.74(c), are GRANTED.

[2] A judgment means a final written decision by the Board, or a termination of a proceeding. 37 C.F.R. § 42.2.

PETITIONER:

Robert Pluta
Amanda Streff
Bryan Nese
MAYER BROWN LLP
rpluta@mayerbrown.com
astreff@mayerbrown.com
bnese@mayerbrown.com

PATENT OWNER:

William Meunier
Michael Renaud
Adam Rizk
MINTZ, LEVIN, COHEN, FERRIS, GLOVSKY AND POPEO, P.C.
wameunier@mintz.com
mtrenaud@mintz.com
arizk@mintz.com

AO 120 (Rev. 08/10)

TO:	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK
Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Delaware on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
	1/23/2017	Delaware

PLAINTIFF	DEFENDANT
Advanced Micro Devices, Inc. and ATI Technologies ULC	LG Electronics, Inc., LG Electronics U.S.A., Inc., LG Electronics MobileComm U.S.A., Inc.

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	US 7,633,506	12/15/2009	ATI Technologies ULC
2	US 7,796,133	9/14/2010	ATI Technologies ULC
3	US 8,760,454	6/24/2014	ATI Technologies ULC
4			
5			

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director **Copy 3—Upon termination of action, mail this copy to Director**
Copy 2—Upon filing document adding patent(s), mail this copy to Director **Copy 4—Case file copy**

AO 120 (Rev. 08/10)

TO:	Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Eastern District of Texas, Marshall Division on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
2:22-cv-00134	5/5/2022	Eastern District of Texas, Marshall Division

PLAINTIFF	DEFENDANT
ADVANCED MICRO DEVICES, INC.; and ATI TECHNOLOGIES ULC	TCL INDUSTRIES HOLDINGS CO., LTD.; TCL INDUSTRIES HOLDINGS (H.K.) LIMITED; TCL ELECTRONICS HOLDINGS LIMITED; TCL TECHNOLOGY GROUP CORPORATION; [illegible]

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	7,742,053	6/22/2010	ATI Technologies ULC
2	8,760,454	6/24/2014	ATI Technologies ULC
3	11,184,628	11/23/2021	ATI Technologies ULC
4	8,468,547	6/18/2013	Advanced Micro Devices Inc., ATI Technologies ULC
5	8,854,381	10/7/2014	Advanced Micro Devices Inc.

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director **Copy 3—Upon termination of action, mail this copy to Director**
Copy 2—Upon filing document adding patent(s), mail this copy to Director **Copy 4—Case file copy**