

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TCL INDUSTRIES HOLDINGS CO., LTD.,

Petitioners

v.

ATI TECHNOLOGIES ULC,

Patent Owner.

PTAB Case No. IPR2024-00366

Patent No. 8,760,454 B2

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 8,760,454**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	1
II. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8(B)	1
A. REAL PARTY IN INTEREST	1
B. RELATED MATTERS	2
C. NOTICE OF COUNSEL AND SERVICE INFORMATION	3
III. PAYMENT OF FEES	4
IV. STANDING	4
V. THE BOARD SHOULD NOT EXERCISE ITS DISCRETION TO DENY INSTITUTION	4
VI. REQUEST FOR <i>INTER PARTES</i> REVIEW OF CLAIMS 1-11 OF THE '454 PATENT	6
A. 37 C.F.R. § 42.104(b)(1): Claims for Which IPR is Requested	6
B. 37 C.F.R. § 42.104(b)(2): Identification of Prior Art and Asserted Grounds for Which IPR is Requested	6
C. Level of Ordinary Skill in the Art	8
D. 37 C.F.R. § 42.104(b)(5): Evidence Supporting Challenge	8
VII. THERE EXISTS A REASONABLE LIKELIHOOD THAT THE CHALLENGED CLAIMS ARE UNPATENTABLE	9
A. Technology Background	9
B. Description of the Alleged Invention of the '454 Patent	13
C. Prosecution History of the '454 Patent and Alleged Priority of Invention	15
D. Ground #1: Claims 1-11 based on the Lindholm Patents	18
1. The Lindholm '685 Patent	18
2. The Lindholm '913 Patent	21
3. Combining the Lindholm Patents	23

TABLE OF CONTENTS
(continued)

	Page
4. Independent Claim 2	23
5. Independent Claim 3	32
6. Independent Claim 4	38
7. Independent Claim 5	39
8. Dependent Claim 6.....	40
9. Dependent Claims 7 and 10	40
10. Dependent Claim 8.....	41
11. Dependent Claim 9.....	42
12. Independent Claim 11	42
13. Independent Claim 1	43
E. Ground #2: Claims 1-11 based on the Combination of Amanatides and Kohn	44
1. Amanatides.....	44
2. Kohn.....	46
3. Combining Amanatides and Kohn	47
4. Independent Claim 2	49
5. Independent Claim 3	57
6. Independent Claim 4	61
7. Independent Claim 5	62
8. Dependent Claim 6.....	62
9. Dependent Claims 7 and 10	63
10. Dependent Claim 8.....	64
11. Dependent Claim 9.....	65
12. Independent Claim 11	65
13. Independent Claim 1	65
F. Ground #3: Claims 1-11 based on the Combination of Selzer and Fiske.....	66

TABLE OF CONTENTS
(continued)

	Page
1. Selzer.....	66
2. Fiske	67
3. Combining Selzer and Fiske.....	68
4. Independent Claim 2	71
5. Independent Claim 3	78
6. Independent Claim 4	80
7. Independent Claim 5	80
8. Dependent Claim 6.....	81
9. Dependent Claims 7 and 10	81
10. Dependent Claim 8.....	83
11. Dependent Claim 9.....	83
12. Independent Claim 11	83
13. Independent Claim 1	84
VIII. CONCLUSION.....	85

PETITIONER’S EXHIBIT LIST

Exhibit #	Reference Name
1001	U.S. Patent 8,760,454 to Morein et al.
1002	Prosecution History of U.S. Patent 8,760,454
1003	Declaration of Hanspeter Pfister, Ph.D.
1004	Curriculum Vitae of Hanspeter Pfister, Ph.D.
1005	U.S. Patent 7,038,685 to Lindholm et al. (“Lindholm ’685”)
1006	U.S. Patent No. 7,015,913 to Lindholm et al. (“Lindholm ’913”)
1007	John Amanatides and Edward Szurkowski, <i>A Simple, Flexible, Parallel Graphics Architecture</i> , In Proceedings of Graphics Interface at 155-160 (Canadian Information Processing Society 1993) published in Proc. Graphics Interface ’93 in May 1993 (“Amanatides”)
1008	Les Kohn and Neal Margulis, <i>Introducing the Intel i860 64-bit Microprocessor</i> , IEEE, Volume 9, Issue 4, pages 15-30, August 1989 (“Kohn”)
1009	Harald Selzer, <i>Dynamic Load Balancing within a High Performance Graphics System</i> , In Proceedings of Rendering, Visualization and Rasterization Hardware (Eurographics' 91 Workshop) at 37-53 (Springer-Verlag 1993) published in 1993 (“Selzer”) [Library of Congress]
1010	Stuart Fiske and William J. Dally, <i>Thread prioritization: A Thread Scheduling Mechanism for Multiple-Context Parallel Processors</i> , In Proceedings of First Symposium on High-Performance Computer Architecture, 1995 at 210-221 (IEEE 1995) published in 1995 (“Fiske”)
1011	IEEE Xplore bibliography page for Fiske, <i>Thread prioritization: A Thread Scheduling Mechanism for Multiple-Context Parallel Processors</i> , visited on May 11, 2023
1012	Declaration of Gordon MacPherson re Authentication of Fiske (May 11, 2023)

1013	IEEE Xplore bibliography page for Kohn, <i>Introducing the Intel i860 64-bit Microprocessor</i> visited on May 11, 2023
1014	Declaration of Gordon MacPherson re Authentication of Kohn (May 11, 2023)
1015	Initial Determination on Violation of Section 337, <i>Certain Consumer Electronics and Display Devices With Graphics Processing and Graphics Processing Units Therein</i> , 337-TA-932 (October 9, 2015)
1016	Harald Selzer, <i>Dynamic Load Balancing within a High Performance Graphics System</i> , In Proceedings of Rendering, Visualization and Rasterization Hardware (Eurographics' 91 Workshop) at 37-53 (Springer-Verlag 1993) published in 1993 (“Selzer”) [University of California, Berkeley, Library]

I. INTRODUCTION

Pursuant to 35 U.S.C. §§ 311-319 and 37 C.F.R. §§ 42.1-.80 & 42.100-.123 *et seq.*, Petitioner respectfully requests *inter partes* review of claims 1-11 of U.S. Patent No. 8,760,454. The challenged claims are unpatentable under 35 U.S.C. §103 over the prior art publications identified and applied in this Petition.

ATI Technologies ULC (“ATI”) did not invent the first “unified shader,” which the ’454 patent discloses as a graphics processor that uses a single processor unit to perform both vertex and pixel operations. Nor did ATI invent the first unified shader that could be flexibly allocated to pixels or vertices based on workload. All of this was accomplished years earlier by Lindholm, Amanatides, Intel, Selzer, and others.

II. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8(B)

Pursuant to 37 C.F.R. § 42.8, Petitioner provides the following mandatory disclosures:

A. REAL PARTY IN INTEREST

Petitioner TCL Industries Holdings Co., Ltd. (“TCL”) and TCL Industries Holdings (H.K.) Limited; TCL Electronics Holdings Limited; TCL Technology Group Corporation; TTE Corporation; TCL Holdings (BVI) Limited; TCL King Electrical Appliances (Huizhou) Co., Ltd.; Shenzhen TCL New Technology Co., Ltd.; TCL MOKA International Limited; TCL Smart Device (Vietnam) Co., Ltd.;

Manufacturas Avanzadas SA de CV; TCL Electronics Mexico, S de RL de CV; TCL Overseas Marketing Ltd., and TTE Technology, Inc. are the real parties-in-interest.

B. RELATED MATTERS

Pursuant to 37 C.F.R. § 42.8(b)(2), Petitioner identifies that the '454 Patent is currently subject to *inter partes* review in *Realtek Semiconductor Corp. v. ATI Technologies ULC*, No. IPR2023-00922, which was instituted on December 1, 2023 and remains pending. Petitioner files herewith a motion for joinder of the instant Petition with the petition in IPRR2023-00922.

Petitioner identifies that the '454 Patent is the subject of a patent infringement lawsuit brought by the Patent Owner, ATI Technologies ULC, against Petitioner in the following cases that may affect or be affected by a decision in this proceeding:

- *Advanced Micro Devices, Inc. et al. v. TCL Industries Holdings Co., LTD., et al.*, Case No. 2:22-cv-00134, now pending in the Eastern District of Texas. This action is stayed “until final resolution of Investigation No. 337-TA-1318.” (Dkt. 65)

The '454 Patent was previously asserted in *Certain Graphics Systems, Components Thereof, and Digital Televisions Containing The Same*, Inv. No. 337-TA-1318, pending in the U.S. International Trade Commission, but the asserted

patent claims were terminated by Order No. 10 on July 14, 2022, upon motion of ATI. The target date for completion of the investigation is January 23, 2024.

The '454 Patent was previously the subject of:

- IPR2017-01225, which was terminated prior to an institution decision;
- *Certain Graphics Systems, Components Thereof, and Consumer Products Containing Same*, Inv. No. 337-TA-1044, but the investigation as to the asserted claims was terminated prior to a determination on the merits; and
- Case No. 1:17-cv-00065 in the District of Delaware, which was dismissed October 18, 2017.

C. NOTICE OF COUNSEL AND SERVICE INFORMATION

Lead counsel: John Schnurer, Reg. No. 52, 196.

Back-up counsel: Yun (Louise) Lu, Reg. No. 72,766; Kevin J. Patariu, Reg. No. 63,210; Kyle R. Canavera, Reg. No. 72,167.

These attorneys can be reached by mail at Perkins Coie LLP, 11452 El Camino Real, Suite 300, San Diego, CA 92130; by phone at (858) 720-5700; and by fax at (858) 720-5799.

Petitioners consent to electronic service. All services and communications to the attorneys listed above may be sent to: [PerkinsServiceTCL-](#)

AMDIPRs@perkinscoie.com. A Power of Attorney is being filed concurrently.

III. PAYMENT OF FEES

The fees pursuant to 37 C.F.R. §42.103 are being paid, and the Patent Trial and Appeal Board (“the Board”) is hereby authorized to charge any further fees to Perkins Coie, LLP: Deposit Account 50-0665.

IV. STANDING

Pursuant to 37 C.F.R. §42.104(a), Petitioner certifies that (1) the ’454 Patent is available for *inter partes* review; and (2) Petitioner is not barred or estopped from requesting *inter partes* review of claims 1-11 of the ’454 Patent on the grounds identified in this Petition. This Petition is timely filed under 35 U.S.C. § 315(b), at least because of the concurrent request for joinder under § 315(c).

V. THE BOARD SHOULD NOT EXERCISE ITS DISCRETION TO DENY INSTITUTION

Although the ’454 Patent is asserted in a case in the Eastern District of Texas, there is no basis to deny institution under §314. This proceeding is still in its very early stages. Petitioner has not filed an answer. The action has been stayed *at the request of the Patent Owner* “until final resolution of Investigation No. 337-TA-1318.” Accordingly, there is no effective trial date and none will be set until sometime after January 23, 2024, the target date for completion of the Investigation, at the earliest. Petitioner filed this Petition and the concurrent

motion for joinder within one month of institution of IPR2023-00922 as required by 37 CFR §42.122. Petitioner will join that ongoing proceeding and does not expect to seek any modification to the schedule already entered in that proceeding. Thus, the first, second, third, fifth and sixth *Fintiv* factors strongly favor institution.

The Board also should exercise its discretion to institute under §325(d) because the Examiner neither cited nor discussed the references that form the second and third grounds for unpatentability. Moreover, Patent Owner did not address the Examiner's rejection based on the '685 patent on its merits and was not able to overcome the rejection on its merits. And as discussed below, ATI's evidence of prior invention was found inadequate in the 337-TA-923 ITC Investigation.

Finally, the Board should not exercise its discretion to deny institution under §314(a) based on IPR2017-01225. The prior petition was filed April 3, 2017, by a different petitioner (LG), about six years before Petitioner was accused of infringing the '454 patent and is alleged to have notice of ATI's claims of infringement. *See Unified Patents Inc. v. Bradium Technologies LLC*, IPR2018-00952, Institution Decision at 15-24 (December 20, 2018).

VI. REQUEST FOR *INTER PARTES* REVIEW OF CLAIMS 1-11 OF THE '454 PATENT

A. 37 C.F.R. § 42.104(b)(1): Claims for Which IPR is Requested

Petitioner requests that the Board finds unpatentable Claims 1-11 of the '454 Patent (the "Challenged Claims"). Such relief is justified as the alleged invention of the '454 Patent was described by others prior to the effective filing date of the '454 Patent.

B. 37 C.F.R. § 42.104(b)(2): Identification of Prior Art and Asserted Grounds for Which IPR is Requested

The prior art relied upon in this Petition is:

U.S. Patent No. 7,038,685 ("685 patent") was filed on June 30, 2003 and issued on May 2, 2006. EX1005. It is therefore prior art to the '454 patent under at least 35 U.S.C. § 102(e).

U.S. Patent No. 7,015,913 ("913 patent") was filed on June 27, 2003 and issued on March 21, 2006. EX1006. It is therefore prior art to the '454 patent under at least 35 U.S.C. § 102(e).

John Amanatides and Edward Szurkowski, *A Simple, Flexible, Parallel Graphics Architecture*, In Proceedings of Graphics Interface at 155-160 (Canadian Information Processing Society 1993) ("Amanatides") was published in Proc. Graphics Interface '93 in May 1993. EX1007. It is therefore prior art to the '454 Patent under at least 35 U.S.C. § 102(a) and (b).

Les Kohn and Neal Margulis, *Introducing the Intel i860 64-bit Microprocessor*, IEEE, Volume 9, Issue 4, pages 15-30, August 1989 (“Kohn”). EX1008, EX1013, EX1014. It is therefore prior art to the ’454 patent under at least 35 U.S.C. § 102(a) and (b).

Harald Selzer, *Dynamic Load Balancing within a High Performance Graphics System*, In Proceedings of Rendering, Visualization and Rasterization Hardware (Eurographics' 91 Workshop) at 37-53 (Springer-Verlag 1993) (“Selzer”) was published in 1993. EX1009, EX1016. It is therefore prior art to the ’454 patent under at least 35 U.S.C. § 102(a) and (b).

Stuart Fiske and William J. Dally, *Thread Prioritization: A Thread Scheduling Mechanism for Multiple-Context Parallel Processors*, In Proceedings of First Symposium on High-Performance Computer Architecture, 1995 at 210-221 (IEEE 1995) (“Fiske”) was published in 1995. EX1010, EX1011, EX1012. It is therefore prior art to the ’454 patent under at least 35 U.S.C. § 102(a) and (b).

The one-year time bar under pre-AIA 35 U.S.C. §102(b) is measured from the effective U.S. filing date of the ’454 Patent, which is no earlier than November 20, 2003.

Inter partes review is requested on the following grounds:

Ground 1: The Challenged Claims are rendered obvious under 35 U.S.C. § 103(a) by the ’685 patent in combination with the ’913 patent.

Ground 2: The Challenged Claims are rendered obvious under 35

U.S.C. § 103(a) by Amanatides in combination with Kohn.

Ground 3: The Challenged Claims are rendered obvious under 35

U.S.C. § 103(a) by Selzer in combination with Fiske.

C. Level of Ordinary Skill in the Art

A person of ordinary skill in the field, at the time the '454 patent was effectively filed, would have had at least a four-year degree in electrical engineering, computer engineering, computer science, or a related field and two years relevant experience in the graphics processing field including developing, designing, or programming hardware for graphics processing units.

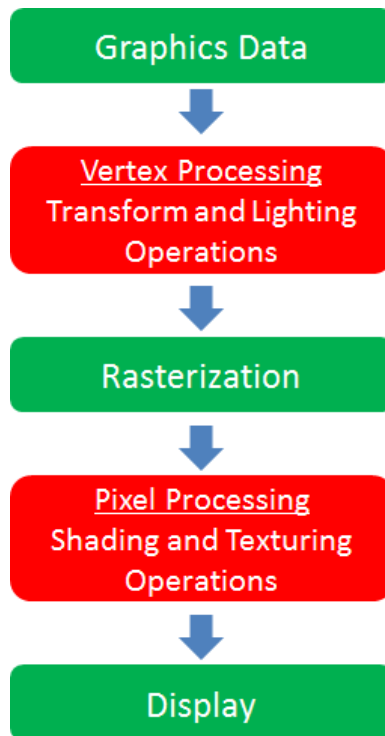
D. 37 C.F.R. § 42.104(b)(5): Evidence Supporting Challenge

The Declaration of Dr. Hanspeter Pfister, Ph.D. (EX1003) and other supporting evidence in the Exhibit List are filed herewith. Dr. Pfister's background and qualifications, and the information provided to him, are discussed in EX1004.

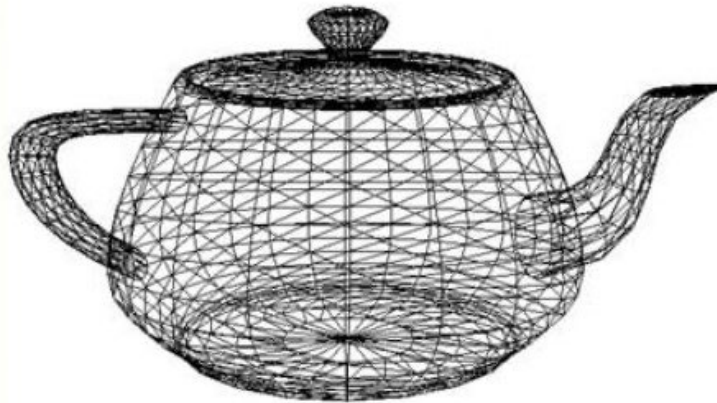
VII. THERE EXISTS A REASONABLE LIKELIHOOD THAT THE CHALLENGED CLAIMS ARE UNPATENTABLE

A. Technology Background

The '454 patent is directed to the typical functions of a graphics processor, which is used to generate complex shapes and structures to be displayed on a screen. EX1001 at 1:38-48; EX1003 ¶37. A graphics processor converts a 3D object or scene (comprised of points in 3D space called “vertices”) into a 2D image to be displayed on a computer screen (comprised of “pixels”). *Id.* Generally, 3D graphics processing starts with creating a mathematical model of each object. *Id.* The model is then processed through a series of steps, referred to as a “graphics processing pipeline,” that renders the scene as a 2D image on a display:



In most cases, 3D objects are conceptualized as a series of triangles (called “primitives”) that cover the surface of an object, such as a teapot:

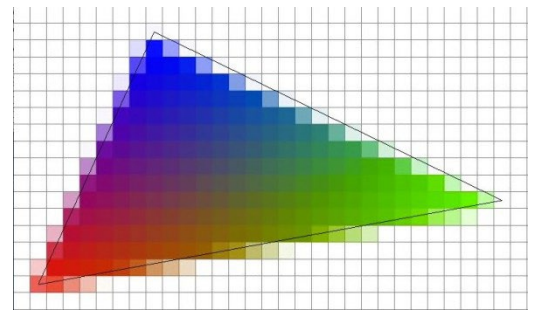


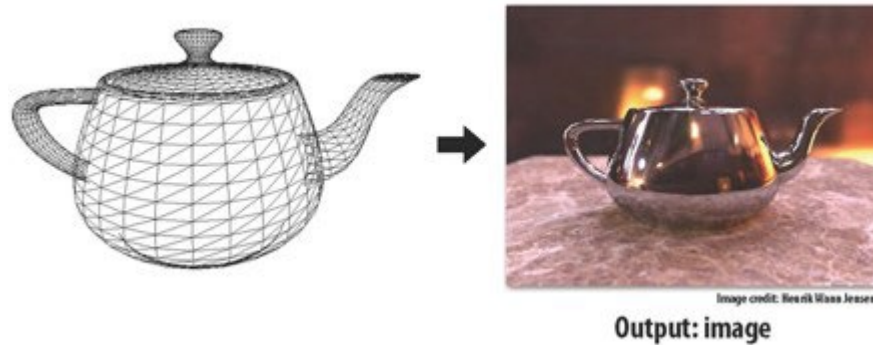
EX1003 ¶38.

Each point of the primitive is called a “vertex,” and each vertex has

certain properties, which are represented as data. *Id.* For example, a vertex includes not just its location, but may also include other information, such as the color of the object and its material properties (*e.g.*, whether it is reflective). EX1003 ¶¶39-40. A vertex processor (the “front end” of the graphics pipeline) transforms these vertices from 3D space into 2D space and determines how lighting and other conditions in the 3D scene impact the color of the vertices. *Id.*

After the vertex processing, a step called rasterization determines what pixels on the 2D screen are covered by each primitive (at right). *Id.* ¶¶41-45. At least one “fragment” is generated for each pixel on the screen (as a result, the terms “fragment” and “pixel” are sometimes used interchangeably). *Id.* The initial values of each pixel (such as its color) are calculated using the vertex data and interpolation. *Id.* A pixel processor (the “back end” of the graphics pipeline) then performs additional operations on the pixels, which includes determining the final color of each pixel. *Id.* All of this information is gathered together for the final image to be displayed on a screen. *Id.*





Conventionally, separate dedicated vertex processors and pixel processors (sometimes referred to as “shaders”) were used to process vertex data and pixel data. EX1003 ¶46. Before the ’454 patent was filed, inventors such as Erik Lindholm, John Amanatides, and Harald Selzer recognized that the conventional approach of using separate and dedicated vertex and pixel processors created inefficiencies. Separate, fixed function vertex and pixel processors could not adapt to the varying needs of each image, some of which have a lot of vertex data to process while others have more pixel data. *Id.* at ¶¶47-48. This was inefficient, as either vertex processors or pixel processors were often idle while the other type of processor became backlogged. *Id.*

What was needed was a way to process both vertex and pixel data through a single piece of hardware that could process either data type and then prioritize or balance a common set of resources between the two

sample types. The inventor of the '685 and '913 Patents, Erik Lindholm, solved the problem. Mr. Lindholm invented a *multithreaded* processor unit that could process *both* vertex and pixel graphics data (sometimes called a “unified shader”) depending on available resources and priorities.

Id. ¶48. The invention also allowed the multithreaded processor unit to dynamically balance or control the number of threads processing vertices or pixels during operation, depending on the available resources.

Amanatides and Selzer similarly invented unified shaders that could process vertices and pixels in the same processing units based on workload, available resources, and processing priorities.

B. Description of the Alleged Invention of the '454 Patent

Similar to the Lindholm patents, the '454 patent notes that one drawback to separate vertex and pixel shader hardware was the increased footprint required on the processor. EX1003 ¶¶49-50. Thus, the '454 patent uses the same piece of hardware, which it also terms a “unified shader,” to run both vertex shader programs and pixel shader programs. EX1001 at 2:58-3:3. The patent defines “unified shader” as a shader that is “configured to perform both vertex and pixel operations.” *Id.* at 3:10-12. Figure 4 of the '454 patent is illustrative of the unified shader.

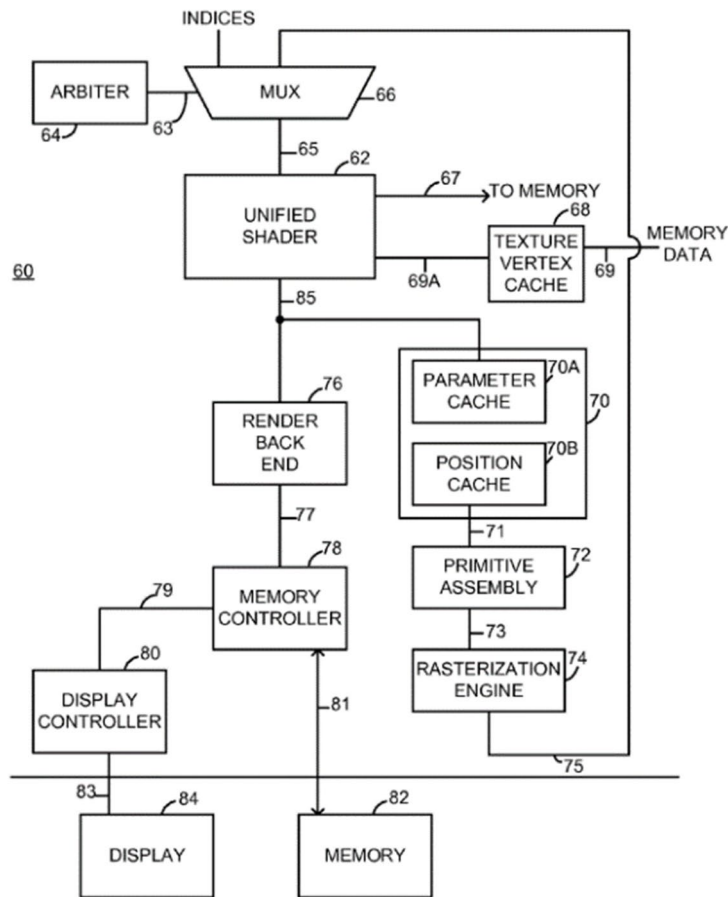


FIG. 4A

Within the unified shader is a general-purpose register block for storing vertex and pixel data to be processed. *Id.* at 4:29-39. The specification discloses that there was nothing inventive about the hardware comprising the “unified shader” itself, which is described as being “a processor (e.g., CPU) 96.” *Id.* at 4:30-34. In one embodiment, the processor 96 is “logically partitioned into two sections,” one of which is

configured to perform 32-bit floating point arithmetic operations and another of which is configured to perform scalar operations (e.g., log, exponent, reciprocal square root). *Id.* It was known to a POSITA that vertex processing typically required floating point operations and pixel processing required scalar operations. EX1003 ¶¶54-55. The processor unit within the unified shader can process vertex data concurrently with pixel data and alternate between the two types of operations. EX1001 at 5:32-36; 6:36-41. Within the unified shader, the unit that maintains instructions for performing vertex and pixel operations is called the sequencer. *Id.* at 4:52-5:5. An arbiter circuit is used to determine which of a plurality of inputs, such as vertices or pixels, are selected for processing by the unified shader. *Id.* at 4:13-28. This determination might be made by giving priority to vertex processing based on whether the general-purpose registers have enough available space to store incoming vertex data, and if not, pixel operations are continued. *Id.* at 5:41-52.

C. Prosecution History of the '454 Patent and Alleged Priority of Invention

The application that led to the '454 patent was filed on May 17, 2011, and issued on June 14, 2014. The '454 patent claims priority to an application filed November 20, 2003. During prosecution, the Examiner

rejected the claims of the '454 patent three times as anticipated by the Lindholm '685 patent. EX1002. ATI never addressed the Lindholm rejection on its merits and was not able to overcome the rejection on its merits. *Id.* Instead, ATI filed an inventor declaration under 37 CFR 1.131, which the Examiner found sufficient in an Office Action dated August 8, 2013, without substantive analysis.

However, ATI's evidence of prior invention was subsequently found insufficient to antedate the '685 patent after a full adjudication on the merits. In *Certain Consumer Electronics and Display Devices with Graphics Processing and Graphics Processing Units Therein*, Judge Pender found that ATI was unable to prove conception prior to the effective filing date of the '685 patent, June 30, 2003, or actual reduction to practice. Inv. No. 337-TA-932, Final Initial Determination (October 9, 2015) (EX1015). For example, Judge Pender rejected ATI's evidence of earlier conception of claim limitations directed to the processing of graphics data by assigning a priority to processing vertices or pixels, as well as the allocation of threads to vertices and pixels based on such a priority. *Id.* at 140-145. Judge Pender also found that ATI's evidence could not support an earlier invention date for load balancing, *i.e.*, determining whether to process pixel data or vertex data based on workload. *Id.* Judge Pender further found ATI's documentary

evidence of prior conception of its “sequencer” to be non-enabling. *Id.* In addition, Judge Pender found ATI’s evidence insufficient to show actual reduction to practice, *i.e.*, that the relevant features disclosed in the early R400 development documents were actually implemented in the later Xenos chip. *Id.* at 145-149.¹

As Judge Pender already adjudicated, ATI cannot prove prior invention as to the features claimed in the ’454 patent. Petitioner anticipates the evidence will fail to antedate the ’685 patent or the ’913 patent because ATI cannot show prior conception, actual reduction to practice, and sufficient diligence in reduction to practice as to the claims of the ’454 patent.

¹ In a later decision, the Federal Circuit held that ATI met its burden of antedating the ’685 patent, but only as to claims of three *other* patents—not the ’454 patent. Those patents did not claim a priority for processing vertices or pixels, load balancing between processing vertices and pixels, and other limitations of the challenged claims of the ’454 patent. *ATI Technologies ULC v. Iancu*, 920 F.3d 1362, 1366-68 (Fed. Cir. 2019).

D. Ground #1: Claims 1-11 based on the Lindholm Patents

1. The Lindholm '685 Patent

The '685 patent describes the same unified shader architecture claimed in the '454 patent and discloses numerous ways to optimize the allocation of resources when simultaneously performing vertex and pixel operations. EX1003 ¶71.

Figure 1 of the '685 patent depicts the basic components of a graphics processor that includes a programmable graphics processing pipeline, which can be programmed to operate on graphics data called “samples,” such as vertices and primitives. *Id.* at 3:59-64. The programmable graphics processing pipeline includes one or more *execution pipelines* as shown in Figure 2:

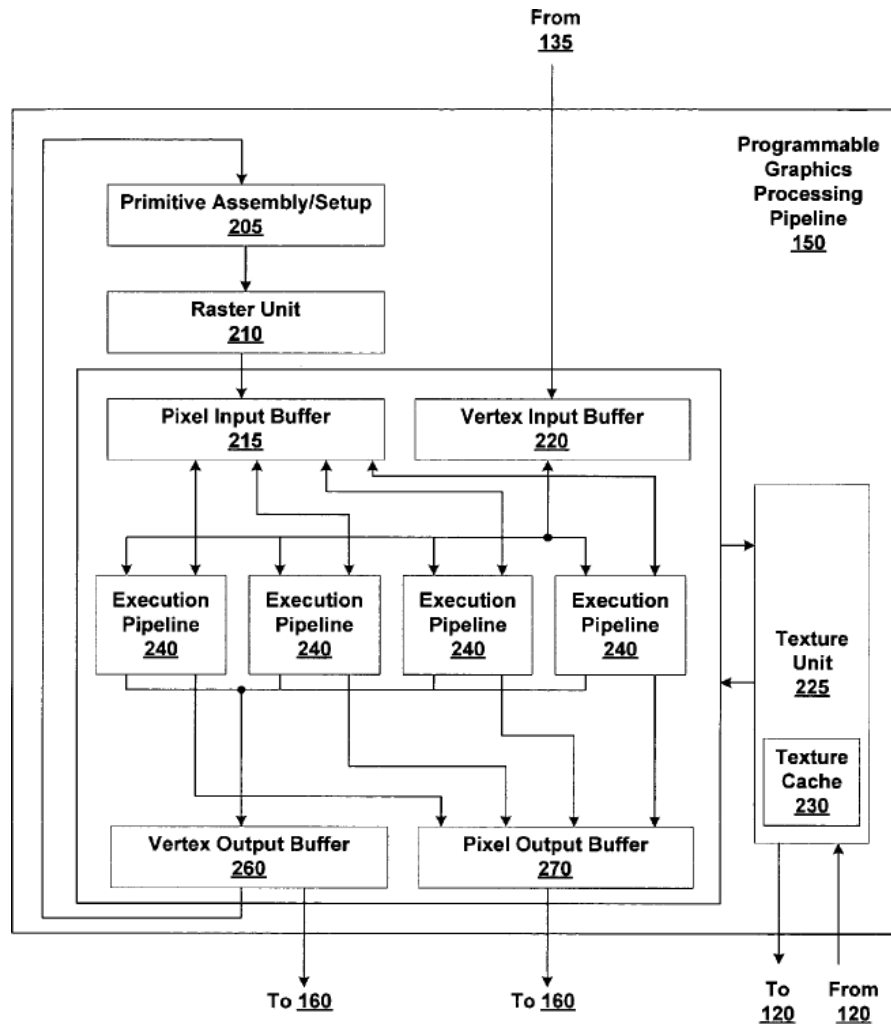


FIG. 2

EX1005 at Fig. 2 and 4:36-41.

Each execution pipeline contains one or more *multithreaded processing units*. EX1005 at 4:27-29. Each multithreaded processing unit accepts samples for processing when a thread is available, sending a signal to the Vertex Input Buffer or to the Pixel Input Buffer when a vertex or a pixel can be accepted for processing. *Id.* at 4:31-36, 5:23-25. Figure 4 shows certain details of the multithreaded processing unit:

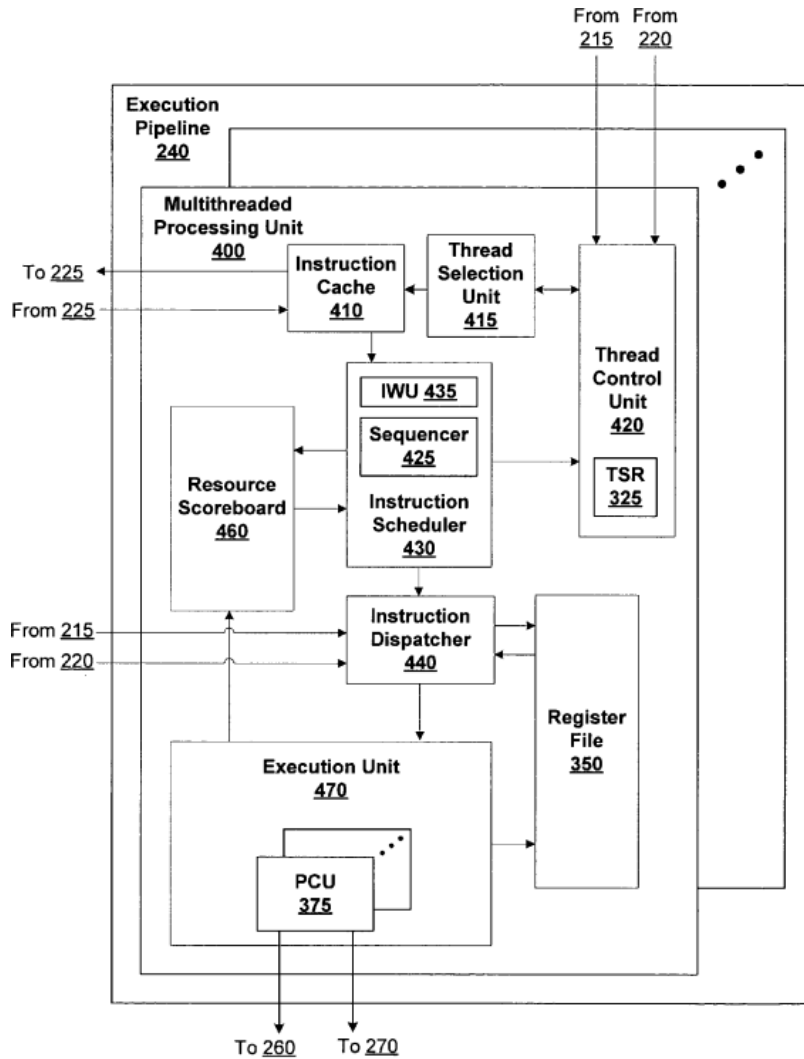


FIG. 4

Id. at Fig. 4.

Each thread can process any sample type, such as a vertex or a pixel (fragment), which enables the multithreaded processing unit to simultaneously process vertices and pixels. *Id.* at 6:52-59; 9:39-49; claims 1, 13, 23, 25, 36, 41, 43. The Thread Control Unit uses a ***thread allocation priority*** to determine whether to assign a vertex or pixel to an available

thread, which priority may be fixed, programmable, or dynamic. EX1005 at 6:40-7:36. After samples are assigned to threads, the Thread Selection Unit uses a “*thread execution priority* assigned to each thread type” (e.g., vertex or pixel) to determine the order of processing. *Id.* at 7:56-8:32. The thread execution priority may also be fixed, programmable, or dynamic. *Id.* at 7:60-61. *See* EX1003 ¶¶72-77.

2. The Lindholm ’913 Patent

The Lindholm ’913 patent was filed just three days before the Lindholm ’685 patent and is directed to the same unified shader architecture. EX1003 ¶¶78-79. The ’913 patent describes how the unified shader can quickly switch between instructions of different threads *while* processing vertex and pixel threads simultaneously, including switching between unfinished vertex threads and unfinished pixel threads. *Id.* ¶79. This allows the unified shader to keep its processing resources fully busy, even when one thread stalls due to the unavailability of source data. *Id.* ¶¶80-81. In the multithreaded unified shader of the ’913 patent, the execution of each thread can proceed independent of any other threads.” EX1006 at Abstract. The ’913 Patent invented a way of tracking source data and then scheduling each program instruction within a particular thread based on whether or not its specified source data (e.g., vertex or pixel data) is available. EX1003 ¶81.

When one thread is waiting for source data needed to execute one of its program instructions, an instruction from another thread can be scheduled for execution instead, which necessarily involves switching between the instruction of one unfinished thread (such as a vertex thread) and an instruction of a different finished or unfinished thread (such as a pixel thread). *Id.*

The components of this novel graphics processor included a multithreaded processing unit, a scheduler, a resource tracking unit, and a dispatcher, as well as additional components such as a thread control buffer and an instruction cache. EX1003 ¶¶82-83. A ***scheduler*** receives program instructions that will process the vertices or pixels. EX1006 at 4:29-32, 5:54-58, 6:63-7:5, 16:48-52, 17:41-44. The scheduler determines whether each program instruction can be executed based on the availability of source data specified by that instruction. *Id.* at 8:49-54. Thus, the multithreaded processing unit can quickly switch back and forth between instructions, and threads (including unfinished threads), of different sample types, as needed in order to prevent any stalls in the pipeline. EX1003 ¶¶83-85. An ***instruction dispatcher*** gathers the specified source data and outputs the program instruction with its source data to the execution unit for processing the corresponding sample. EX1006 at 9:40-42; 16:55-57; 17:47-50. *See*

EX1003 ¶¶84-85.

3. Combining the Lindholm Patents

Both the '685 and the '913 patents share the same inventor (Erik Lindholm), the patents were filed within days of each other, the patents share much of the same specification and figures, and the patents are directed to the same multithreaded unified shader. EX1003 ¶86. A POSITA at the time of the '454 patent's alleged invention would have recognized the Lindholm patents were disclosing different details of the same unified shader architecture. *Id.* Moreover, a POSITA would have been motivated to look to the '913 patent for the additional details that it disclosed regarding the multithreaded processing unit of the '685 patent, particularly regarding the scheduling of threads for execution independent of any other threads. This improved performance because a stalled thread does not prevent the execution of other threads. *Id.* Thus, a POSITA would have been specifically motivated to combine the teachings of the two Lindholm patents. *Id.*

4. Independent Claim 2

Independent claim 2 recites, with brackets added for later reference:

2. A unified shader, comprising:

[a] a general purpose register block for maintaining data;

- [b] a processor unit;
- [c] 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
- [d] 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.

(a) Claim 2 [preamble] – Unified Shader

To the extent the preamble is a limitation, the Lindholm Patents each disclose a unified shader. EX1003 ¶¶88-91.

Within the graphics processor of the '685 patent, EX1005 at Fig. 1, there is a programmable graphics processing pipeline, which includes a plurality of execution pipelines, each of which can receive vertex data from the vertex input buffer 220 and pixel data from the pixel input buffer 215, as shown in Fig. 2:

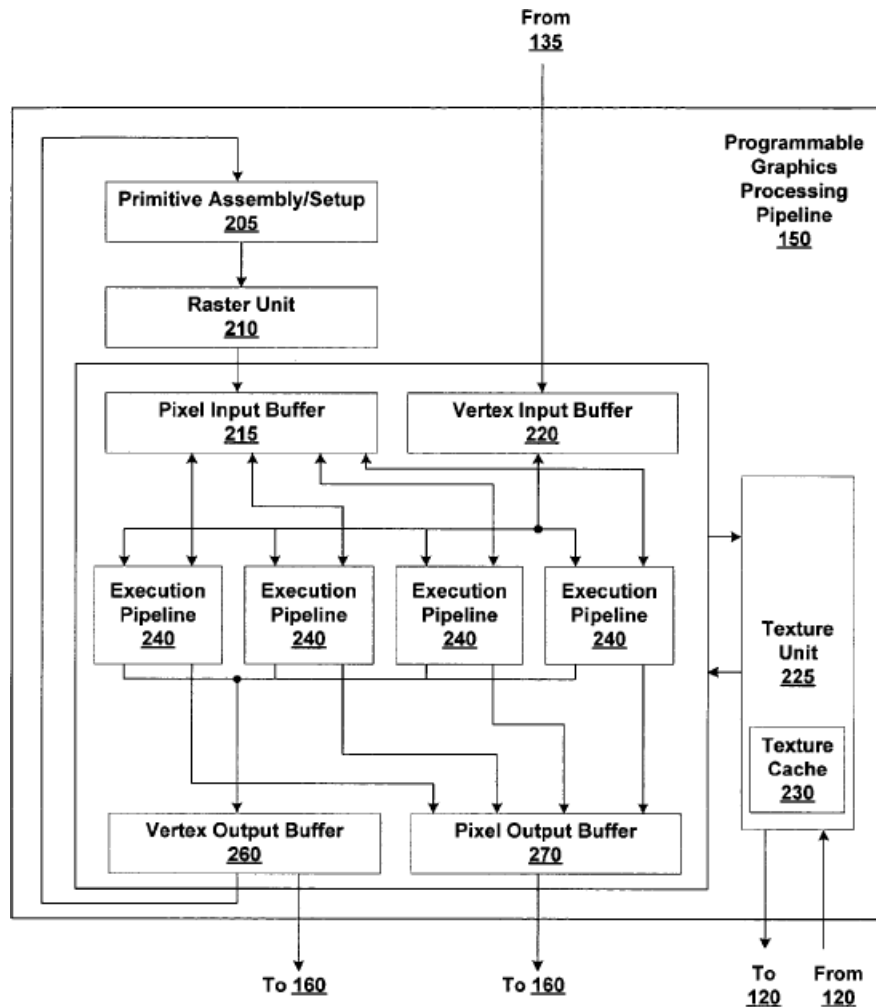


FIG. 2

EX1005 at Fig. 2, 4:22-5:35. *See also* EX1006 at Fig. 2, 4:35-5:45; EX1003 ¶89. “Programmable Graphics Processing Pipeline 150 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample or any other data.” EX1005 at 3:57-64; 4:13-41.

Each execution pipeline can perform vertex operations, such as transforming vertices from “object space” to another coordinate system.

EX1005 at 4:42-5:10. Each execution pipeline can also perform pixel operations, such as “texture mapping, shading, blending, and the like.” *Id.* at 5:11-49.

Within each execution pipeline are one or more multithreaded processing units, as shown in Figure 4:

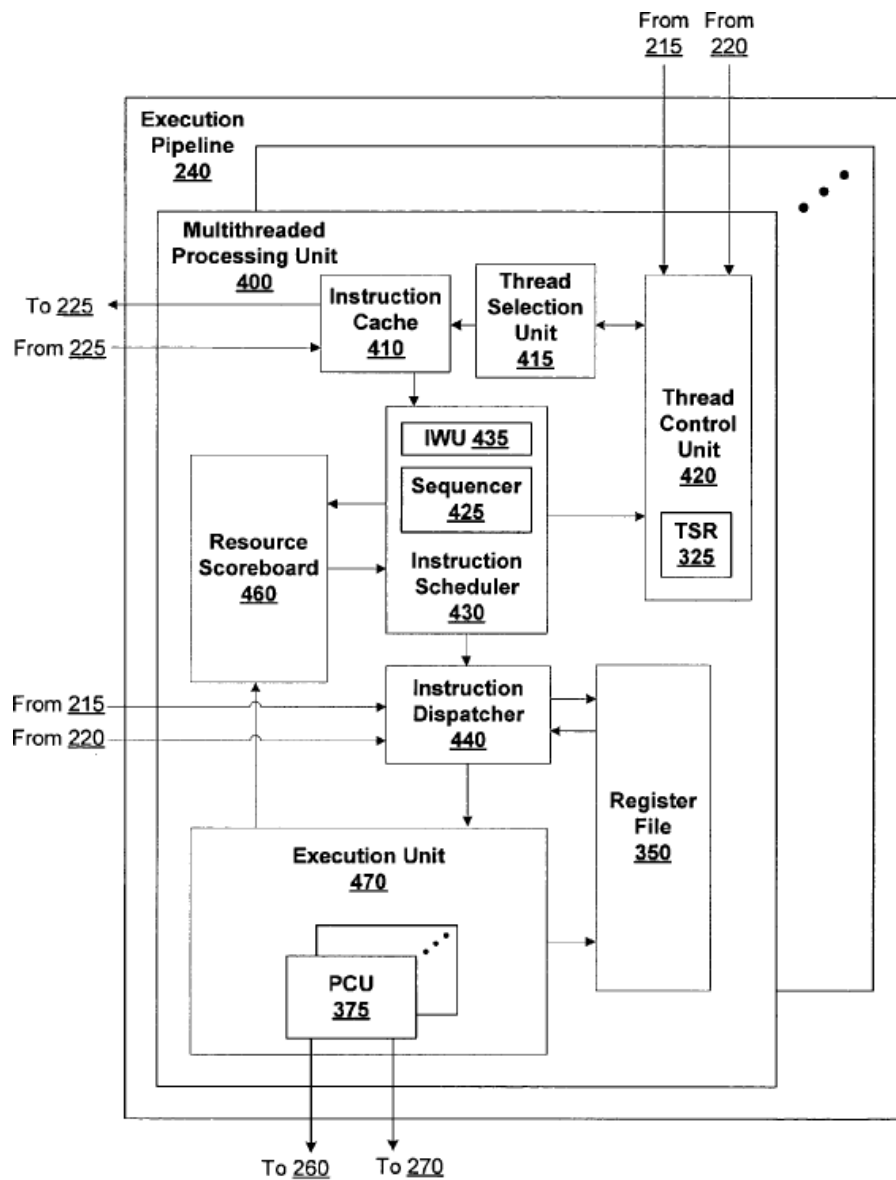


FIG. 4

EX1005 at Fig. 4, 4:27-29; 5:36-49, 8:30-32; EX1006 at Fig. 4, 4:40-42, 8:19-20;

EX1003 ¶91. Each multithreaded processing unit can simultaneously process

vertex data and pixel data (as described further with respect to claim elements 2[b]

and [d]). EX1005 at 6:38-49. *See also* EX1006 at 7:19-42; EX1003 ¶¶91. The Programmable Graphics Pipeline 150, or alternatively, each Execution Pipeline 240 (and/or components of each of the foregoing) in each of the Lindholm Patents corresponds to the “unified shader” of the ’454 patent.

(b) Claim 2[a] – General Purpose Register Block

The Lindholm Patents each disclose “a general purpose register block for maintaining data,” which includes both vertex data and pixel data. Register File 350 maintains source data, including vertex data and pixel data. “In addition to Register File 350, other locations of source data include Pixel Input Buffer 215, Vertex Input Buffer 220, locations in Local Memory 140, locations in Host Memory 112, and the like.” EX1005 at 5:50-6:3. *See also id.* at 7:37-43, 8:47-58; EX1003 ¶¶92.

The Thread Control Unit 420 receives vertex data from the Vertex Input Buffer and pixel data from the Pixel Input Buffer. *Id.* at 5:50-6:3. The Pixel Input Buffer and the Vertex Input Buffer also include “a register file, FIFO ... cache, or the like” and a POSITA would have understood these to be portions of a general purpose register block. EX1005 at 4:22-26, 5:32-35; EX1003 ¶¶93-94. Accordingly, each of these components contain registers that can maintain both vertex and pixel data, and individually or collectively constitute “a general purpose register block” as required by

claims 1 and 2.

(c) Claim 2[b] and [d] – Processor Unit

Claim elements 2[b] and [d] are disclosed by each of the Lindholm Patents. EX1003 ¶¶95-100. Within each multithreaded processing unit, “Execution Unit 470 can simultaneously process samples of different types [e.g., vertices and pixels], and likewise, execute threads of different types.” EX1005 at 9:39-49; claims 1, 13, 23, 25, 36, 41, 43. The Execution Units and/or the programmable computation units (PCUs) that comprise each Execution Unit satisfy the “processor unit” limitations of claim 2, as well as the processor unit limitations in claims 3-11 and the “processor” of claim 1.

Each Execution Unit 470 and its PCUs 375 execute instructions that perform pixel operations (such as instructions that generate a pixel color) and instructions that perform vertex operations (such as instructions that generate vertex position and appearance data). EX1003 ¶96. *See* EX1005 at 4:22-5:49, 9:33-49, 11:1-14, 15:27-35. Based on the disclosures of each of the Lindholm Patents, a POSITA would have recognized that the Execution Units/PCUs “execute instructions that generate a pixel color in response to selected data.” EX1003 ¶97. For example, the ’685 patent describes how the PCUs “perform operations such as tessellation, perspective correction, texture mapping, shading, blending, and the like” in response to receiving

pixel samples from the Pixel Input Buffer. EX1005 at 5:23-35. A POSITA would have recognized that such operations generate a pixel color. EX1003 ¶97. *See also* '454 Patent at 1:38-48 and 2:8-22 (explaining how the pixel shading part of graphics processing provides the color value for each pixel of a rendered object).

Similarly, a POSITA would have recognized that the Execution Units/PCUs “execute instructions that generate vertex position and appearance data in response to selected data.” EX1003 ¶98. It was well-known in the art that operations performed on vertices impacted their position and appearance. *Id.* For example, the '685 patent discloses that the execution pipelines, when operating on vertex samples, would “transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system,” thereby generating vertex position, and apply texture maps, which generates appearance data, in response to receiving vertex samples. *Id.*; EX1005 at 4:42-64, 9:33-49.

Execution Units 470 and PCUs 375 are operatively coupled to Register File 350, as well as the other memory locations discussed in the Lindholm Patents, *supra* claim element 2[a], which comprise the general purpose register block, and receive vertex and pixel data therefrom. *See* EX1003 ¶99. EX1005, FIGS. 1, 4.

The vertex data and pixel data received by the Execution Units 470 and PCUs 375 is “selected” for processing by the Thread Selection Unit, as described further below in relation to the “arbiter” of claim 10, according to thread allocation priorities and execution priorities. *Id.* at 6:61-7:36, 7:56-8:32; EX1003 ¶100; *see infra* at claim elements 3 [c], 4[c], 11[a].

(d) Claim 2[c] - Sequencer

The Lindholm Patents each disclose the claimed “sequencer,” which corresponds to the combination of the Instruction Cache 410, the Instruction Scheduler 430 (which includes Sequencer 425 and IWU 435), and the Instruction Dispatcher 440. EX1003 ¶¶101-102. These components function together to maintain instructions operative to cause the Execution Unit/PCUs to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block, *supra* claim 2[d]. *Id.*

The Instruction Cache 410 maintains and outputs program instructions to the Instruction Scheduler 430. EX1005 at 8:33-46; EX1003 ¶102. “Instruction Scheduler 430 contains storage resources to store a predetermined number of instructions” and evaluates whether instructions can be executed by the Execution Unit/PCUs depending on the availability of computation resources and source data in Register File 350. *Id.* at 8:47-

9:4. “When Instruction Scheduler 430 determines which instructions and associated threads will be executed,” it processes instructions using Sequencer 425. *Id.* at 9:5-32. “Instruction Dispatcher 440 [then] 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.” *Id.* at 9:33-49.

Accordingly, the “sequencer” is coupled to the “processor unit” (Execution Unit/PCUs) and to the general purpose register block (Register File 350 and the Vertex/Pixel Input Buffers). *See id.* Fig. 4.

5. Independent Claim 3

Independent claim 3 recites, with brackets added for later reference:

3. A unified shader comprising:
 - [a] a processor unit operative to perform vertex calculation operations and pixel calculation operations; and
 - [b] shared resources, operatively coupled to the processor unit;
 - [c] 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.

(a) Claim 3 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 3[a] and [c] – Processor Unit

Each of the Lindholm Patents disclose Execution Units/PCUs as the claimed “processor unit,” each of which perform both vertex calculation operations and pixel calculation operations. *See supra* at §2[b] and [d]. Each of the Lindholm Patents disclose that the Execution Units/PCUs are operative to perform either vertex calculation operations or pixel calculation operations based on a priority, which may be a thread allocation priority or a thread execution priority, and which may be fixed, programmable, or dynamic. EX1003 ¶¶105-19.

Because vertex operations and pixel operations occur simultaneously within each multithreaded processing unit, data is assigned to available threads based on a “thread allocation priority” that is specified for each sample type (vertex or pixel) and “may be fixed, programmable, or dynamic.” EX1005 at 6:60-7:36; EX1006 at 7:43-58. Vertex or pixel threads are then selected for processing by the Thread Selection Unit using an “execution priority” that is specified for each sample type assigned to the threads (*e.g.*, vertex and pixel) which priority “may be fixed, programmable or dynamic.” EX1005 at 7:56-64; EX1003 ¶106.

With a fixed vertex thread allocation priority, vertex calculation operations would be performed until enough “shared resources” become available, and then pixel calculation operations would be performed.

EX1003 ¶107-08. “In one embodiment the thread allocation priority may be fixed, always giving priority to allocating vertex threads and pixel threads are only allocated if vertex samples are not available for assignment to a thread.” EX1005 at 6:60-7:5. Conversely, a fixed pixel thread allocation priority would give priority to processing pixel threads, and vertex operations would only be performed if no pixel samples were available for thread assignment. *Id. See also id.* at 10:17-54 (describing Fig. 5B, application of thread allocation priority), 16:10-55 (describing Fig. 9B, application of thread execution priority).

In other examples, a priority can be applied to flexibly control the Execution Units/PCUs to perform vertex or pixel operations based on vertex or pixel workload. EX1003 ¶109. For example, if a thread allocation priority is set to prioritize pixel operations, priority will be given to pixel operations based on the amount of pixel samples waiting to be processed. “Dynamically modifying a thread allocation priority for pixel samples based on the amount of sample data in Pixel Input Buffer 215 permits Pixel Input Buffer 215 to drain faster.” *Id.* at 7:6-36. “For example, a dynamically

determined thread allocation priority may be determined based on a number of ‘pending’ pixels.” *Id.* Conversely, a thread allocation priority that prioritizes vertices will prioritize vertex operations based on the amount of vertices waiting to be processed. *Id.*

Alternatively, the thread allocation priority may give priority to either vertex workload or pixel workload, depending on which is greater. EX1003 ¶110. “In another embodiment, Thread Selection Unit 415 is configured to read thread entries based on the amount of sample data in Pixel Input Buffer 215 and the amount of sample data in Vertex Input Buffer 220.” *Id.* at 7:65-8:14. *See also id.* at 15:42-57 (thread execution priority determined for pixel or vertex, “based on the amount of sample data in Pixel Input Buffer 215 and the amount of sample data in Vertex Input Buffer 220”).

The priority for processing vertices or pixels can be set or changed based on the availability of resources, including space available in the “general purpose register block” or “store,” Register File 350. EX1003 ¶111-12. “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,” which includes data from either the “Pixel Input Buffer 215 [or] Vertex Input Buffer 220.” EX1005 at 7:56- 64; 8:47-

64; 9:2-4; 9:50-62; EX1006 at 8:47-64. Thread allocation or execution decisions can also be based on space available in Register File 350, or “the amount of storage for intermediate data generated during processing of a sample.” *Id.*, and at 7:37-40, 12:9-36, 16:19-24.

Since vertex and pixel operations are conducted simultaneously, the Lindholm patents further explain how the allocation of processing resources between vertex and pixel operations can change during operation. *Id.* at 12:9-27. *See also* EX1006 at 7:6-12; EX1003 ¶113. Thus, the Execution Units/PCUs are able to perform vertex operations and pixel operations at various degrees of completion by switching between instructions in the Instruction Cache and Instruction Scheduler. EX1003 ¶113.

Although none of the claims of the ’454 patent requires a multithreaded processor or switching between “unfinished” pixel threads and “unfinished” vertex threads, this is also disclosed by the Lindholm Patents. This is one way in which the Lindholm Patents disclose freely switching between vertex and pixel operations at varying degrees of completion. *See generally* EX1006 at 7:19-10:3; 13:17-14:62; EX1003 ¶114. The ’913 patent discloses instruction-level scheduling and execution process starts with the Instruction Scheduler, which can output multiple instructions per clock cycle corresponding to multiple threads. EX1006 at

8:21-46, 9:24-30; EX1003 at ¶¶114-15. The '913 patent further describes how the execution units are able to simultaneously process multiple samples based on the Instruction Scheduler and Instruction Dispatcher issuing instructions from multiple threads (i.e., freely switching between instructions of different threads and thread types) stored in the multithreaded processing unit. EX1006 at 13:24-39; EX1003 ¶¶115-16. In this manner, the execution units of the Lindholm multithreaded processor are able to switch back and forth between vertex operations and pixel operations. *Id.* ¶¶117-18. Each execution unit may have multiple threads in operation simultaneously (at least at different stages of its pipeline) and it is also able to switch between vertex and pixel threads to ensure that the graphics processor is dynamically load balanced. *Id.* ¶119. Accordingly, the processing units as claimed in claims 1, 3, 4, 5 and 11 are disclosed or rendered obvious by the '685 and '913 patents, which disclose simultaneously processing vertices and pixels based on workload and/or availability of resources, and at various degrees of completion.

(c) Claim 3[b] – Shared Resources

Petitioner incorporates by reference the discussion *supra* at §2[a].

The “shared resources”² include the “general purpose register block,” (’454 patent at 5:32-52), which in the ’685 patent includes Register File 350, as well as the Pixel Input Buffer 215 and Vertex Input Buffer 220. *Id.*; EX1003 ¶120. Each are “coupled to” the Execution Unit/PCUs through Instruction Dispatcher 440, which gathers vertex and pixel data from those memory locations and outputs it to the Execution Units/PCUs, and thus the Execution Units/PCUs are “operative to use the shared resources for either vertex data or pixel information.” *Id.*; EX1005 at Fig. 4, 9:33-49.

6. Independent Claim 4

Independent claim 4 recites, with brackets added for later reference:

4. A unified shader comprising:
 - [a] a processor unit operative to perform vertex calculation operations and pixel calculation operations; and
 - [b] shared resources, operatively coupled to the processor unit;
 - [c] 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.

² “Shared resources” are understood to refer to components that store both vertex and pixel data.

(a) Claim 4 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 4[a] and [c] – Processor Unit

See supra at §3[a] and [c].

(c) Claim 4[b] – Shared Resources

See supra at §3[b].

7. Independent Claim 5

Independent claim 5 recites, with brackets added for later reference:

5. A unified shader comprising:

[a] a processor unit;

[b] 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.

(a) Claim 5 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 5[a] – Processor Unit

See supra at §2[b].

(c) Claim 5[b] – Sequencer

See supra at §§2[c], 3[a] and [c].

8. Dependent Claim 6

Dependent claim 6 recites, “The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.” As explained *supra* at §2[c], the “sequencer” in the ’685 patent includes Instruction Dispatcher 440, circuitry operative to fetch data from a memory. EX1003 ¶129. “Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 ... and outputs the instruction and source data to Execution Unit 470.” EX1005 at 9:33-49, 11:9-14, 15:24-30.

9. Dependent Claims 7 and 10

Dependent claim 7 recites, “The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.” Dependent claim 10 recites, “The shader of claim 7, wherein the control signal is provided by an arbiter.”

The ’685 patent discloses the Thread Selection Unit 415 as the claimed “selection circuit” and Thread Control Unit 420 as the claimed “arbiter.” EX1003 ¶131. The Thread Selection Unit functions in response to control signals provided by the Thread Control Unit 420, as it reads thread entries (including thread state data) from the Thread Control Unit, which may assign threads or samples to threads based on thread allocation

priority or execution priority. *Id.*; EX1005 at 6:60-7:5, 7:6-8:17; 15:14-17:7. Moreover, the “Thread Selection Unit 415 is configured to read thread entries based on the thread execution priority assigned to each thread type” or “based on the amount of sample data in Pixel Input Buffer 215 and the amount of sample data in Vertex Input Buffer 220,” which signals are provided by the Thread Control Unit. *Id.* and at 7:56-8:14, 16:10-55, Fig. 3-4 (thread execution priority register shown as part of Thread Control Unit). To the extent not expressly disclosed, it would have been obvious to a POSITA to provide the priority control signal from the Thread Control Unit to the Thread Selection Unit. EX1003 ¶131. The Thread Selection Unit outputs thread entries and other thread data, such as state data, which provides information to the “store” (Register File 350) at least through the threads selected for execution, using the Instruction Cache, Instruction Scheduler, and Instruction Dispatcher, as well as the Execution Unit. EX1005 at 8:15-9:67. Moreover, information (including intermediate data) is provided to Register File 350 as a result of the threads selected for execution. *Id.* at 5:50-6:3, 7:37-40, 8:47-9:4, 9:33-66.

10. Dependent Claim 8

Dependent claim 8 recites, “The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to

the selected data.” This functionality is discussed *supra* at §2[d].

11. Dependent Claim 9

Dependent claim 9 recites, “The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected data.” This functionality is discussed *supra* at §2[d].

12. Independent Claim 11

Independent claim 11 recites, with brackets added for later reference:

11. A unified shader comprising:

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

[b] an instruction store and

[c] 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.

(a) Claim 11 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 11[a] and [c] – Processor Unit

See supra at §§2[c], [d], 3[a], [c].

(c) Claim 11[b] – Instruction Store

The “instruction store” is met by the same components that satisfy the

“sequencer” limitation of claim 2. *See supra* at §§2[c], [d], 3[a], [c].

13. Independent Claim 1

Independent claim 1 recites, with brackets added for later reference:

1. A method carried out by a unified shader comprising:

[a] 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

[b] 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.

Claim 1 is a method claim that puts the limitations of apparatus claim 3 into method claim format. For example, the steps to be performed in claim 1 align with the functional elements of the processor unit, and the ’685 patent discloses giving vertex operations priority as long as there is available space in the general-purpose register block. EX1003 ¶140. The “instruction store” is met for the same reasons as the “sequencer” of claim 2. Accordingly, claim 1 is met and rendered obvious for the same reasons as claims 2 and 3. *See supra* at Claims 2 and 3.

E. Ground #2: Claims 1-11 based on the Combination of Amanatides and Kohn

The '454 patent does not disclose nor claim a multithreaded processor. EX1003 ¶¶143, 159-64. Rather, it claims, at best, a pipelined processor that can execute multiple instructions concurrently with respect to different data types (e.g., vertex and pixel). *Id.* Vertex and pixel data may be processed simultaneously because there are separate partitions in the processor for floating point calculations (which are needed for vertex operations) and for scalar/integer calculations (which are used for pixel operations). *Id.*; EX1001 at 3:4-10; 4:40-51. This was nothing new and was invented a decade earlier when Amanatides used the Intel i860 processor to develop a unified shader architecture that could load balance depending on the workload of the processor. *Id.*

1. Amanatides

Amanatides describes a unified shader architecture in a parallel processing graphics system. EX1003 ¶144. Amanatides uses an Intel i860 processor in its graphics processing system, which prioritizes vertex or pixel processing and implements “auto load balancing” its workload between vertex and pixel data. EX1007 at 157; EX1003 ¶144-46.

Vertex and pixel processing in Amanatides occurs in the same

processor unit because “the front and back ends are collapsed onto the same processors.” EX1007 at 157; EX1003 ¶¶147-48. In Amanatides, the “front-end” of the graphics pipeline, vertex processing (also referred to as “Geometry” at the time) is included in the same processor that performs the “back-end” or pixel processing (also referred to as “Rendering” or “Rasterization”). EX1003 ¶148. Thus, Amanatides termed the unified shader unit the G/R unit and illustrated how it combined the separate vertex/geometry processors with the pixel/rendering processors:

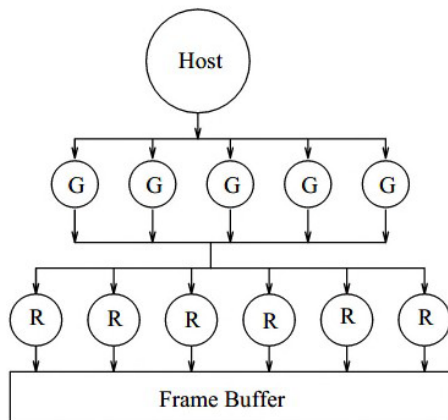


Figure 2

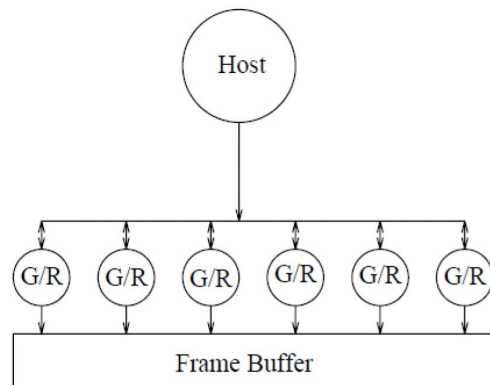


Figure 3

EX1007 at Fig. 2, 3, and 157.

Amanatides specifically focused on inventing a unified graphics processing architecture in order to save space, power, and cost. *Id.*

Amanatides “wanted an architecture that minimized graphics hardware. The result, we believe, is an inexpensive, powerful and flexible system The

stress was on low cost, flexibility and power.” *Id.* at 155-56.

Amanatides recognized the inefficiencies in a graphics processor that uses dedicated vertex and pixel processors:

When performing interactive graphics the bottlenecks in the system would change depending on the types of rendering involved. If there was texture mapping, the pixel nodes were the bottleneck. If only simple shading was performed, the pipe nodes were the bottleneck.

EX1007 at 157; EX1003 ¶150.

Amanatides invented an architecture that could freely switch between vertex and pixel processing in the same processor unit. EX1003 ¶151.

“[T]he design is auto load balancing. For example, if there is a lot of texture mapping, the processors spend most of their time on this. If most of the primitives are simple, then the processors spend the majority of their time on the front end computation.” EX1007 at 158-59; EX1003 ¶¶152-53.

Amanatides further discloses “high and low priority packets,” and the ability to “map” vertex (Geometry) work or pixel (Rendering) work to low priority or high priority packets. *Id.* Low priority packets cannot get onto the bus if high priority packets are waiting. *Id.*

2. Kohn

Kohn describes the Intel i860 processor in detail. EX1008; EX1003

¶154. The Intel i860 processor was divided into “blocks for integer operations, floating- point operations, and instruction and data cache memories.” *Id.* The processor was able to “support multiple operations in each clock cycle” and enabled “balanced performance between integer and floating-point computations.” *Id.* The parallel execution of these data types (floating point and scalar/integer) enabled fast graphics processing. *Id.* at 16-17. This is exactly how the ’454 patent describes the processor unit in its “unified shader.” EX1001 at 3:4-17, 4:40-51.

Because the Intel i860 processor was designed for graphics processing, it could efficiently process both vertex and pixel data. EX1003

¶158. For example, the “floating point hardware of the CPU efficiently performs the transformation calculation and advanced lighting calculations required for 3D graphics.” EX1007 at 16 (describing vertex operations). In addition, the “use of integer operations for shading and hidden-surface removal” was required for 3D image display. *Id.* (describing pixel operations). Thus, the Intel i860 was capable of simultaneously operating on vertex data (in the floating point units) at the same time as pixel data (in the integer/scalar units). EX1003 ¶158.

3. Combining Amanatides and Kohn

A POSITA would have been motivated to combine Amanatides with

Kohn because both references use the Intel i860 processor and both references describe processing graphics with that same processor. EX1003 ¶165. The Kohn reference simply provides details regarding the same Intel i860 processor that was used in Amanatides. EX1003 ¶165. One of ordinary skill in the art would have been specifically motivated to combine the two references because they are directed to the same processor, and because Kohn provides the i860 processor details that are not expressly set forth in Amanatides. *Id.*

Amanatides states, “After exploring several DSP microprocessors, we decided on the Intel i860. It is a powerful general-purpose microprocessor optimized for graphics.” EX1007 at 4. Kohn states that the designers of the i860 processor “recognized a synergistic opportunity to incorporate a 3D graphics unit that supports interactive visualization of results.” EX1008 at 15. The i860 processor also contains “hardware necessary to implement transparent caching for multiprocessor systems moved costs beyond the silicon budget of this implementation.” *Id.* at 25. A POSITA would have recognized that the i860 processor is ideal to implement the flexible multiprocessor graphics architecture described in Amanatides, based on the disclosure of Amanatides, and would look to the description of the Intel i860 processor in Kohn for additional details of its structure and operation.

EX1003 ¶166.

4. Independent Claim 2

(a) Claim 2 [preamble] – Unified Shader

To the extent the preamble is a limitation, Amanatides alone or in combination with Kohn discloses a graphics processor with a unified shader.

EX1003 ¶¶167-69.

The vertex and pixel processing in Amanatides may occur in the same processor unit:

[T]he front and back ends are collapsed onto the same processors. In this case, the host distributes primitives in a round-robin manner to each of the processors.

Id. at 157; EX1003 ¶169. In Amanatides, the vertex processing (G) is included in the same processor that performs the pixel processing (R). *Id.* Thus, Amanatides termed the unified shader unit the G/R unit. *Id.*

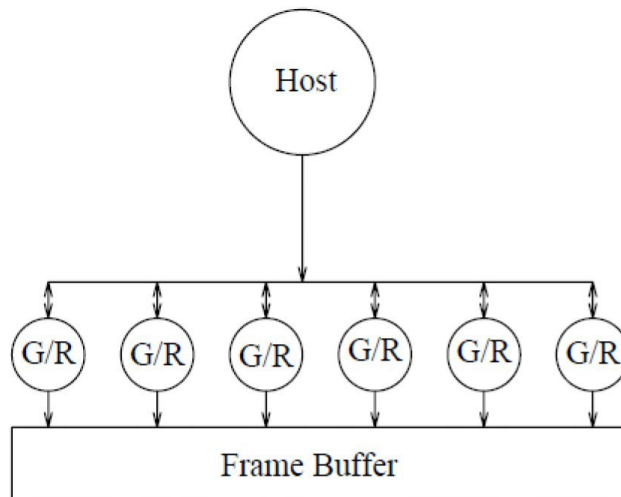


Figure 3

(b) Claim 2[a] – General Purpose Register Block

Amanatides discloses a DRAM that stores both vertex and pixel data and is operatively coupled to the processor unit. EX1007 at 158; EX1003 ¶170. This is also depicted in Figure 4:

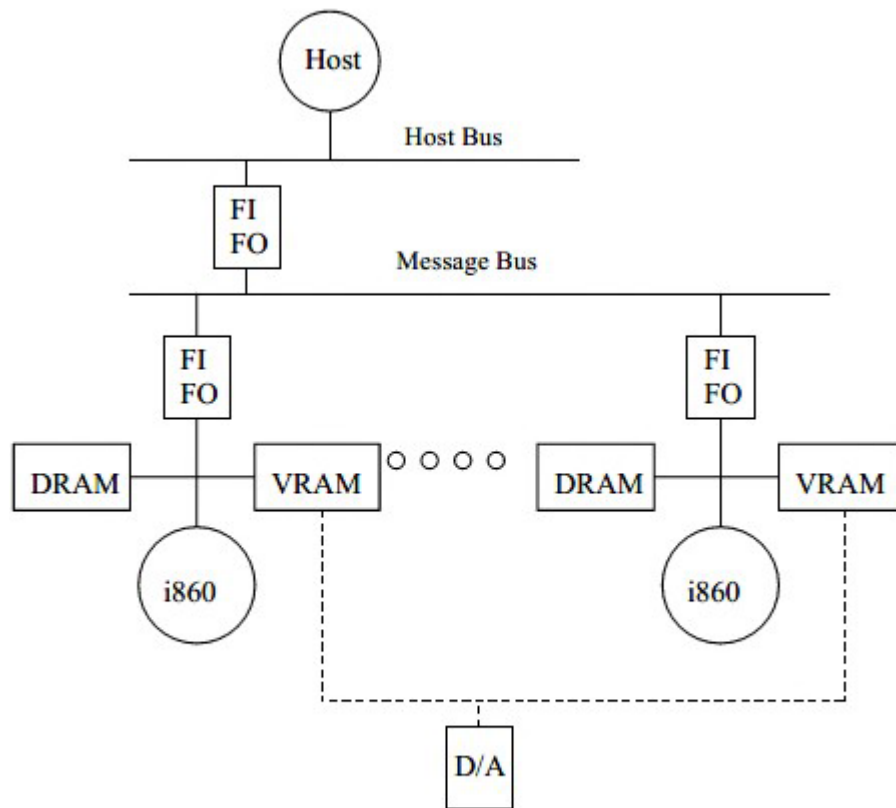


Figure 4

EX1007 at 158. Amanatides also discloses 2 MB of VRAM associated with each i860 processor. EX1007 at 157-58. Amanatides further discloses a bus and an input FIFO for each processor that store both vertex and pixel data:

There are two types of packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. ...

EX1007 at 158-59; EX1003 ¶170.

Vertex data and pixel data are also stored in an on-chip cache within each i860 processor. *Id.* See also EX1008 at 15 (“we divided the chip area

evenly between blocks for integer operations, floating point operations, and instruction and data cache memories”), Fig. 2 (showing the “data cache”). Accordingly, each of these components contain registers that can maintain both vertex and pixel data, and individually or collectively constitute “a general purpose register block” as required by claims 1 and 2 and the data “store” of claim 5.

(c) Claim 2[b] and [d] – Processor Unit

Claim elements 2[b] and [d] are disclosed by each of Amanatides and Kohn and the combination of the references. EX1003 ¶172. The claimed “processor unit” can be read on each G/R shown in Amanatides (the i860 processor). EX1007. Alternatively, the “processor unit” can be read on the functional units within the i860 processor (namely, the floating point control unit, adder unit, and multiplier unit together with the 3D graphics integer unit, collectively or individually, each of which can perform both vertex and pixel operations). EX1008 at 15, 20, 24.

Amanatides discloses that in its G/R processors, the “front end” of graphics processing (vertex or “geometric” operations) and the “back end” (pixel or “rendering” operations) occur simultaneously in the same G/R processor. *Id.* at 156-57; EX1003 ¶173.

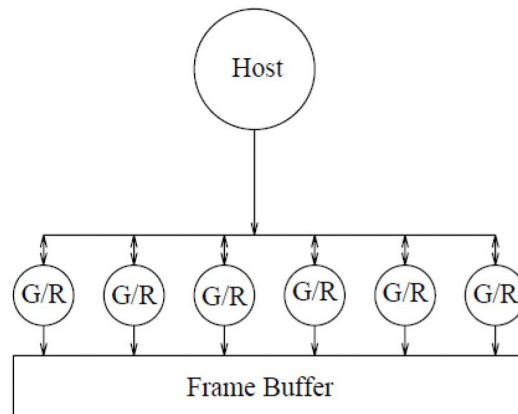


Figure 3

EX1007 at Fig. 3.

The '454 patent describes a pipelined processor architecture with logical partitions that enable the same processor to process vertex and pixel operations. EX1003 ¶174.

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 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 scalar operations (e.g. log, exponent, reciprocal square root) of the unified shader.

EX1001 at 4:37-48. Even though a multithreaded architecture was developed as another way to accomplish simultaneously processing of

multiple data types (*see, e.g.*, the Lindholm Patents), the '454 patent relied upon partitioned sections or stages of a pipelined processor to perform simultaneous processing of vertex and pixel data. EX1003 ¶174.

As explained in Kohn, the i860 processor used in Amanatides as a unified shader could perform parallel execution of these vertex and pixel data types, in the same manner as processor 96 in the '454 patent. EX1008 at 16-17. For example, Kohn describes the floating-point instruction units of the i860 processor as being able to “efficiently perform[] the transformation calculations and advanced lighting calculations required for 3D graphics,” which are vertex operations. EX1008 at 23-24; EX1003 ¶175. But the floating point units also support scalar operations, which as the '454 patent discloses, are typical of pixel operations. EX1001 at 4:29-51; EX1008 at 16, 22-23; EX1003 []. Meanwhile, the “graphics unit” in the i860 could more quickly perform “back-end rendering operations” such as “shading,” which are pixel operations. EX1008 at 24; EX1003 ¶175. But the “graphics unit” could also perform calculations on the vertices during transformation, *i.e.*, vertex operations. EX1008 at 24. In this way, each i860 processor in the Amanatides architecture was capable of “auto load balancing” between vertex and pixel work (G/R). EX1007 at 157-58. As used in Amanatides’ load balancing architecture, the i860 processor became a unified shader that

could switch between vertex and pixel processing depending on the type of data present in the registers. EX1003 ¶175.

Moreover, each of Amanatides and Kohn discloses (and a POSITA would have recognized) that the execution of instructions for geometry (vertex) operations generate position and appearance data and the execution of instructions for rendering (pixel) operations generate the color of a pixel. *See* EX1007 at 157- 59; EX1008 at 24; EX1003 ¶176.

Amanatides also discloses that geometry/vertex operations are performed by its processor units *in response to* receiving “geometry messages” with vertex data (GMs) and that rendering/pixel operations are performed *in response to* receiving “rendering messages” with pixel data (RMs). EX1007 at 159-160; EX1003 ¶177.

(d) Claim 2[c] - Sequencer

Amanatides alone or in combination with Kohn discloses the claimed “sequencer.” Amanatides includes at least an on-chip cache to store program instruction for vertex and pixel calculation operations. EX1007 at 157-58 (“To get back the speed, we wanted the processor to have an on-chip cache.”); EX1003 ¶178. The on-chip cache, together with the Memory Management Unit (EX1008 at 15, 17, 20), comprise the claimed “sequencer” and sequencer “circuitry” of the ’454 patent.

Kohn provides additional detail regarding the on-chip instruction cache, which is shown in Fig. 2. EX1008. Kohn describes its “instruction cache” in detail, including a dual instruction mode. *Id.* at 16, 24-25. Kohn also discloses the ability to issue up to three operations each clock cycle to support parallel execution by the various functional units within the processor. *Id.* The instructions maintained by the sequencer would include instructions operative to perform both vertex and pixel operations, as the unified shader architecture discloses each G/R processor (the i860) performing front-end (vertex) operations and back-end (pixel) operations simultaneously, which operations are necessarily performed by executing instructions on pixel data and vertex data. *See supra* at claim 2[b] and [d]; EX1003 ¶179.

Finally, the “sequencer” is “coupled” to the various memories that comprise the “general purpose register block.” *See supra* at claim 2[a]. For example, the instruction cache is “coupled to” the data cache within the i860 processor, as well as the DRAM, VRAM, FIFO and bus associated with each i860 processor.

EX1007 at Fig. 4; EX1008 at Fig. 2; EX1003 ¶180.

5. Independent Claim 3

(a) Claim 3 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 3[a] and [c] – Processor Unit

Amanatides discloses a multiprocessor system, in which each i860 processor may be considered the claimed “processor unit,” EX1007, or alternatively, the functional units within the i860 processor (*i.e.*, the floating point control unit, RISC integer unit, adder unit, and multiplier unit together with the 3D graphics unit within the i860 processor, individually or collectively). EX1008 at 15, 20, 24. Each of these processor units performs both vertex calculation operations and pixel calculation operations. *See supra* at §2[b] and [d].

Amanatides discloses that its i860 processors and the processor units within those processors are operative to perform either vertex calculation operations or pixel calculation operations based on a priority and/or using automated load balancing. EX1003 ¶¶182-83. Amanatides discloses how its new “minimalist” unified shader architecture combines the vertex processing that comprises the “front end” of the graphics pipeline with the pixel processing of the “back end.” EX1007 at 157. Amanatides then describes several advantages of the system, one of which is that the

processors can spend more of their time on vertex (front end) processing or pixel (back end) processing, depending on how much vertex work or pixel work needs to be done, *i.e.*, is stored in the FIFO, DRAM, VRAM, and on-chip data cache of each processor:

There are several advantages to this approach. First, the design is auto load balancing. For example, if there is a lot of texture mapping, the processors spend most of their time on this. If most of the primitives are simple, then the processors spend the majority of their time on the front end computation.

Id. This discloses load balancing, namely, performing more vertex operations or more pixel operations depending on vertex/pixel workload (*i.e.*, the amount of vertex data or pixel data waiting to be operated on) or the amount of memory available for each type of work.

Amanatides also utilizes a priority allocation system in which priority is given to one of the types of graphics data—either pixel data or vertex data—so that pixels are processed until none remain or until the processor cannot store additional pixels, or alternatively, so that vertices are processed until none remain or until the processor cannot store additional vertices.

EX1003 ¶183. Amanatides did this by controlling when vertex processing tasks (GMs) and pixel processing tasks (RMs) are submitted to the unified

processor units over the bus.

As was mentioned earlier packets on the bus have two priorities, low and high. Low priority packets cannot get onto the bus if high priority packets are waiting. If we map GMs to low priority packets and RMs to high priority packets then the solution is in sight. ...

What will eventually happen is that each processor will oscillate between working on RMs and GMs; in both cases, it will be doing useful work. The long FIFOs (room for 32 packets) make sure that the processors are kept busy.

EX1007 at 158-59. Either GMs or RMs could be assigned “high priority.”

EX1003 ¶184. Moreover, the amount of storage used for each of vertex data and pixel data will be determined based on the priority given to each type of data. In addition, Amanatides discloses that the FIFOs can be implemented in bidirectional FIFO chips, which can be programmed to “raise signals at various levels of filling.” EX1007 at 158; EX1003 ¶185.

Kohn further notes that the floating point and integer processor units both had multiple stages, which means that as different operations are selected to be loaded into the pipeline they may be at different stages of execution. EX1008 at 22-23; EX1003 ¶183. For example, Kohn describes performing operations in parallel, in a “dual-operation mode” within the

“pipelined mode,” and that both floating-point operations and scalar operations could “take advantage of the pipelining capability.” *Id.* A POSITA would have understood, based on the combination of Amanatides and Kohn, that the functional units within each i860 processor could not only be performing vertex operations and pixel operations simultaneously, but at various degrees of completion, based on switching between instructions in the instruction cache. *Id.*

Accordingly, Amanatides discloses “processor units” that prioritize or load balance either vertex operations or pixel operations, at different degrees of completion, and based on the amount of vertex work or pixel work needs to be performed and/or the amount of memory space or shared resources available for each type of processing. EX1003 ¶¶184-185. In this manner, the combination of Amanatides and Kohn, together or in combination, meet or render obvious the processor unit limitations of claim 3, as well as the workload-related limitations of claims 1, 4, 5 and 11, directed to the processor unit, general purpose register block, and sequencer/instruction store.

(c) Claim 3[b] – Shared Resources

Petitioner incorporates by reference the discussion *supra* at §2[a]. The “shared resources” include the “general purpose register block,” (’454

patent at 5:32-52), which in Amanatides can encompass the bus, FIFO, DRAM and VRAM for each i860 processor, *supra* at §2[a], as well as the instruction and data cache, memory management unit, bus control unit, core registers, floating-point registers, buses, and other circuitry within each processor, as shown in Kohn. These resources are all “shared” for both vertex and pixel processing. The bus, FIFO, DRAM and VRAM are all operatively coupled to the i860 processor, and those components together with the on-chip instruction and data cache, memory management unit, bus control unit, core registers, floating-point registers, buses, and other circuitry are operatively coupled to the functional units within the processor (which individually or collectively may also be considered the “processor unit” of the claim). *See supra* at §§2[a], [b] and [c]; EX1003 ¶186.

6. Independent Claim 4

(a) Claim 4 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 4[a] and [c] – Processor Unit

See supra at §3[a] and [c].

(c) Claim 4[b] – Shared Resources

See supra at §3[b].

7. Independent Claim 5

(a) Claim 5 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 5[a] – Processor Unit

See supra at §2[b].

(c) Claim 5[b] – Sequencer

See supra at §§2[c], 3[a] and [c].

8. Dependent Claim 6

Dependent claim 6 recites, “The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.” As explained *supra* at §2[c], the “sequencer” in Amanatides includes the on-chip instruction cache. EX1003 ¶193. As described by Kohn, the instruction cache in the i860 processor is connected to a Memory Management Unit (MMU). EX1008 at Figure 2, p. 17. The MMU translates virtual addresses (used by the software) into physical addresses (used by the hardware). “The address translation process maps virtual address space onto actual address space...” *Id.* at p. 20. The physical addresses are issued to the external memory via a 32-bit External Address bus. The data from the external memory is received via a 64-bit External Data bus via the Bus Control Unit and stored in the Data Cache for processing. *Id.* at Figure 2, p.

17.

Accordingly, the circuitry of the i860 processor as disclosed in the combination of Amanatides and Kohn satisfies the claim limitations of “sequencer” as well as “further ... circuitry operative to fetch data from a memory.”

9. Dependent Claims 7 and 10

Dependent claim 7 recites, “The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.” Dependent claim 10 recites, “The shader of claim 7, wherein the control signal is provided by an arbiter.”

As explained *supra* at claim 3[a] and [c], Amanatides alone or in combination with Kohn discloses or renders obvious an arbiter that determines which input (e.g., vertex or pixel) to provide to the processor units, as well as the claimed selection circuit and control signal. EX1003 ¶194-95. Amanatides utilizes a priority allocation system in which priority is given to one of the types of graphics data (e.g., pixels) so that pixel samples are processed until none remain or until the processor cannot store additional pixel samples. *Id.* Some packets (vertex or pixel) can be assigned a high priority, and other packets assigned a low priority. *Id.* “Low priority packets cannot get onto the bus if high priority packets are waiting.” *Id.*;

EX1007 at 158-59. Accordingly, the “selection circuit” of claim 7 is met by at least the bus and FIFOs, which are operative to provide packets (“information”) to the DRAM, VRAM, and on-chip data cache (“store”) in response to a control signal, such as whether the packets are designated high- priority or low priority as explained further below. EX1003 ¶195.

Assignment of low-priority and high-priority packets to the time slots of the bus is described by Amanatides as “arbitration.” EX1007 at 159. Moreover, the FIFOs can be programmed to “raise signals at various levels of filling.” *Id.* Amanatides describes the packets that are transferred on the message passing bus between processors (“THE BUS”). *Id.* “There are two types of packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs.” *Id.* “Running at 10MHz, 2 time slots are required for arbitration.” *Id.* In that case, the “arbitration” happens on the message passing bus, which would be performed by logic circuits that are not further described in Amanatides, but would be recognized by a POSITA reading Amanatides as being necessarily present. EX1003 ¶196. The “control signal” is part of the packet, *e.g.*, as “low” or “high” priority. *Id.*

10. Dependent Claim 8

Dependent claim 8 recites, “The shader of claim 5, wherein the

processor unit executes instructions that generate a pixel color in response to the selected data.” This functionality is discussed *supra* at §2[d].

11. Dependent Claim 9

Dependent claim 9 recites, “The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected data.” This functionality is discussed *supra* at §2[d].

12. Independent Claim 11

(a) Claim 11 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 11[a] and [c] – Processor Unit

See supra at §§2[c], [d], 3[a], [c].

(c) Claim 11[b] – Instruction Store

The “instruction store” is met by the same components that satisfy the “sequencer” limitation of claim 2. *See supra* at §§2[c], [d], 3[a], [c].

EX1003 ¶201.

13. Independent Claim 1

See supra at Claims 2 and 3. Claim 1 is a method claim that puts the limitations of apparatus claim 3 into method claim format. For example, the steps to be performed in claim 1 align with the functional elements of the processor unit, and Amanatides discloses or at least renders obvious that

vertex operations could be given priority as long as there is sufficient space in the general-purpose register block for vertex processing. EX1003 ¶202. The “instruction store” is met for the same reasons as the “sequencer” of claim 2. Accordingly, claim 1 is met and rendered obvious for the same reasons as claims 2 and 3.

F. Ground #3: Claims 1-11 based on the Combination of Selzer and Fiske

1. Selzer

Selzer is a unified shader graphics processor that uses parallel processing to enable simultaneous execution of multiple types of graphics data. EX1003 at 205-06. Selzer addresses the same problem as the ’454 patent—preventing underutilization of resources when processing vertex and pixel graphics data. *Id.* at 212. In Selzer’s system, “the processing units have to be able to adapt their activities to the actual processing requirement of the scene.” EX1009 at 46. Selzer solves this problem with processing units that can perform processing on different types of graphics data. The Selzer system includes “geometry modules,” each of which “transforms and clips the graphic primitives, subdivides biparametric patches and the lighting calculations that are necessary,” and “rendering modules,” each of which “performs the shading algorithms and transfers pixel data to the

frame buffer.” *Id.* at 40. Importantly, “if required, the rendering modules are capable of performing all the geometry calculations too.” *Id.* at 46-47.

Through its flexible processing system, Selzer achieves dynamic load balancing among different graphics tasks, namely vertex and pixel processing. EX1003 at 207-08. Selzer notes that its system can scale to include multiple graphics processing modules and that “[t]he system is organized as a pipeline with additional parallelism on function level by multiplying geometry and rendering modules.” EX1009 at 43; EX1003 ¶¶209-211. Each of the rendering modules can switch between different graphics tasks, such as vertex and pixel tasks, to achieve load balancing:

To enable such a dynamic load balancing and to speed up geometry calculation dynamically if required, the rendering modules are capable of performing all the geometry calculations too. After rendering an object any processor may run into an idle state if there is no rendering data in his input buffer. Performing a task switch it will request new unprocessed geometry objects and continue geometry calculations. In this way ***an automatic[] load balancing is achieved across all the processors.***

EX1009 at 46-47 (emphasis added).

2. Fiske

Fiske discloses a multithreaded processor that provides register resources to allow for rapid context switching between several threads.

EX1003 ¶213. Fiske’s system is designed to be dynamic and uses a thread prioritization scheme to guide the flexible approach to thread scheduling.

EX1010 at 210-11. Because Fiske provides multiple contexts so that multiple threads can be loaded simultaneously, multiple threads are present and available to run at any time. *Id.*

Fiske notes that scheduling of threads and efficiency of a multithreaded processor is a two-step consideration. EX1003 ¶¶213-14. First, Fiske’s system decides which threads should be loaded into one of the available contexts. *Id.* at 210. Second, out of the threads that are loaded into the context, Fiske’s system employs a thread prioritization scheme to decide which thread has its instructions executed. *Id.* at 211. Fiske notes that if all threads in its system were considered equal—and there was no prioritization used—then runtime performance would suffer. EX1003 ¶¶215-220. Thus, depending on the workload of the processor, different threads can be selected of different types based on the type of priority employed. EX1010 at 210-11.

3. Combining Selzer and Fiske

It would have been obvious for a POSITA to combine the teachings of Selzer and Fiske in order to use the multithreaded architecture of Fiske with the flexible graphics processing system of Selzer. EX1003 ¶¶221-226.

Both Selzer and Fiske are directed dynamic load balancing within a processing system. EX1003 ¶222. The only differences between the approaches of Selzer and Fiske is that Fiske incorporates multiple hardware thread contexts to allow for rapid switching among threads, whereas Selzer utilizes a scalable approach to parallelism by multiplying geometry and rendering modules, while also having modules that can perform both geometry (vertex) calculations and rendering (pixel) calculations. *Id.* ¶¶222-23.

A POSITA would have found it a straight-forward implementation that yielded predictable results to combine the architecture of Selzer with the multithreaded approach of Fiske. EX1003 ¶223. For example, the exemplary configuration of Selzer's architecture is three geometry and five rendering modules. *Id.* Each of these modules is responsible for performing operations on a particular graphics thread (either vertex or pixel). *Id.* Selzer then explains how this architecture, where the rendering modules can perform both vertex and pixel processing, allows for dynamic load balancing because the rendering modules can switch from performing pixel tasks to performing vertex tasks. *Id.* This approach is exactly what Fiske proposes, but Fiske condenses the approach into a single processing module that itself includes multiple thread contexts. *Id.* In other words, a POSITA

would find it obvious to implement Fiske's multithreaded processing architecture as the rendering modules in Selzer. *Id.* A POSITA would have recognized that this combination would perform the exact same operations as Selzer's numerous rendering modules, but would allow for savings in chip space. *Id.*

A POSITA would have further recognized that both Selzer and Fiske are directed to dynamic load balancing schemes and the scheduling approach of either reference could be utilized in the new combination. *Id.*

¶224. For example, using the Selzer scheduling approach, the rendering module would prioritize pixel processing ahead of vertex processing, which would result in designating additional thread contexts to pixel threads and scheduling them first for processing. *Id.* Utilizing Fiske's scheduling approach, the system could dynamically adjust to allow for additional register space for either vertex or pixel threads depending on the workload of the system. *Id.* Then, based upon the number of pixel or vertex thread contexts that are loaded, Fiske's approach would dynamically schedule a thread for execution.

Fiske further explains that any multithreaded architecture would include decision-making functionality for thread assignment/allocation and thread execution, as those decisions must be made in any system. *See*

EX1010 at 210-11. Accordingly, a POSITA taking advantage of the known benefits of the unified shader of Selzer with the additional benefits of a multithreaded architecture would necessarily include such decision-making functionality. See *id.*; EX1003 ¶225.

Thread prioritization was a routine and well-understood way to make the scheduling decision in Fiske. EX1003 ¶226. A POSITA would have understood from Selzer that prioritizing vertex and pixel processing at times is important and thus would have been motivated to use “thread prioritization” for the scheduling decisions. *Id.* Moreover, utilizing “thread prioritization” in a multithreaded system would have been a simple matter of design choice, implementing a known technique in a predictable way. *Id.*

4. Independent Claim 2

(a) Claim 2 [preamble] – Unified Shader

To the extent the preamble is a limitation, Selzer alone or in combination with Fiske discloses a graphics processor with a unified shader. EX1003 ¶227. Selzer describes “a graphics processor architecture with a high degree of parallelism connected to a distributed frame buffer. ... Within the architecture an automatic load balancing mechanism is presented which distributes the processing load between geometry and rendering section.” EX1009 at 37. Selzer’s architecture includes a “rendering

module” that “performs the shading algorithms and transfers pixel data to the frame buffer. The rendering module also supports too all functions of the geometry module.” *Id.* at 40-41. Because the rendering module performs both vertex and pixel operations, it is a unified shader according to the ’454 patent. EX1003 ¶227.

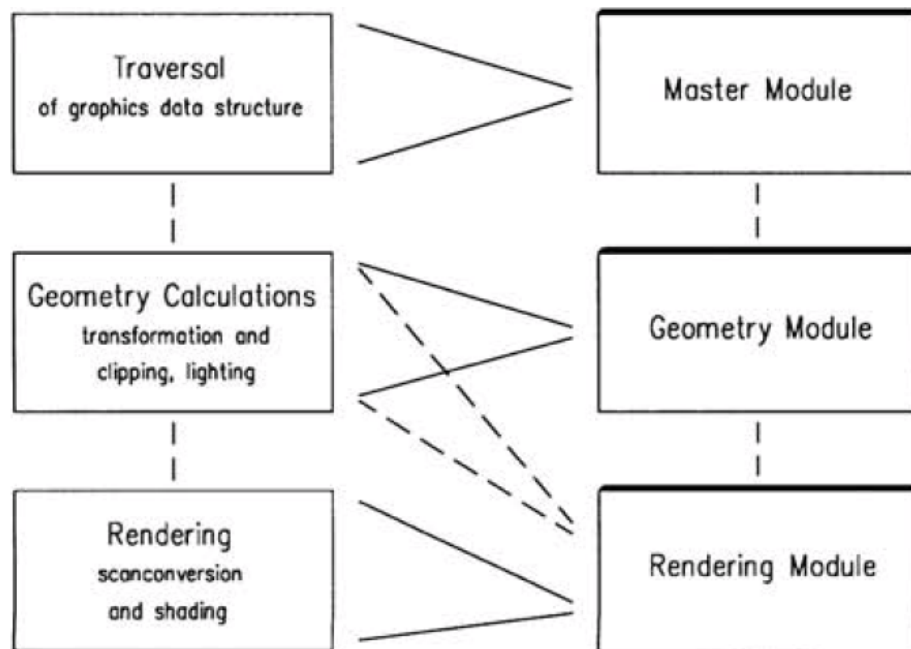


FIGURE 4.1. Mapping functional sections to hardware

As indicated previously, the claims of the ’454 patent do not claim nor require a multithreaded processor unit as the claimed “unified shader.” However, to the extent that limitation is unnecessarily read into the claims, the combination of Selzer and Fiske teaches a multithreaded unified shader. EX1003 ¶228. A POSITA would find it obvious to implement Fiske’s

multithreaded processing architecture as a rendering module in Selzer. *Id.*

As noted above, instead of five rendering modules required in Selzer, utilizing the multiple thread contexts would allow for a single rendering module in Selzer. This combination rendering module with multiple thread contexts (*i.e.*, a multithreaded processor unit) would allow for processing of both vertex and pixel operations and further allow for dynamic scheduling and load balancing of the threads. EX1003.

Accordingly, under any interpretation of claim 2, it is met by the combination of Selzer and Fiske.

(b) Claim 2[a] – General Purpose Register Block

Selzer discloses that each of its rendering processors and the master module includes a block of registers (general register blocks) in the form of programmable memories and FIFOs that receive and maintain all geometry (vertex) and rendering (pixel) data. EX1009 at 44-45; EX1003 ¶229. In addition, a distributed frame buffer is coupled to each rendering unit.

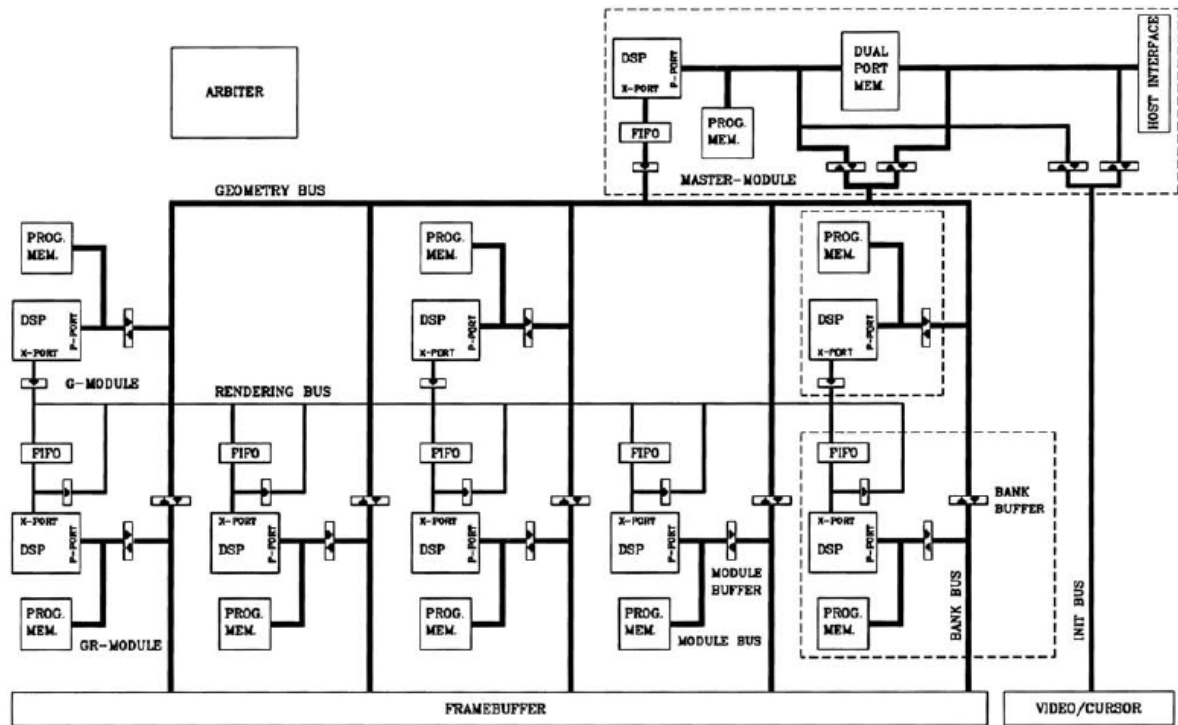


FIGURE 4.4. System architecture

Id. All system busses transfer data to and from the framebuffer and the “master” module for distribution to the rendering modules. *Id.* at 43-44. As Selzer notes, “in the case of synchronizing....the master takes over system control and bypasses the pipeline with a direct access to the appropriate resource.” *Id.* at 41. Thus, the master module may direct either type of graphics data to a rendering module in order to load balance. EX1003 ¶229. Accordingly, Selzer teaches claim element 2[a].

When combined, Selzer and Fiske disclose multiple thread contexts, which Fiske describes as storing the threads that are available for execution. EX1003 ¶230. The thread contexts described by Fiske include the data and

instructions for execution. *Id.* Fiske teaches that these loaded thread contexts are selected by a thread prioritization scheduling scheme and sent to a processor unit. *Id.*

Accordingly, Selzer, either alone or in combination with Fiske, meets claim element 2[a].

(c) Claim 2[b] and [d] – Processor Unit

Selzer, either alone or in combination with Fiske, teaches the claimed processor unit. EX1003 ¶¶232-35. Selzer discloses at least one rendering module that is able to process both vertex and pixel operations. EX1009 at 41-43.

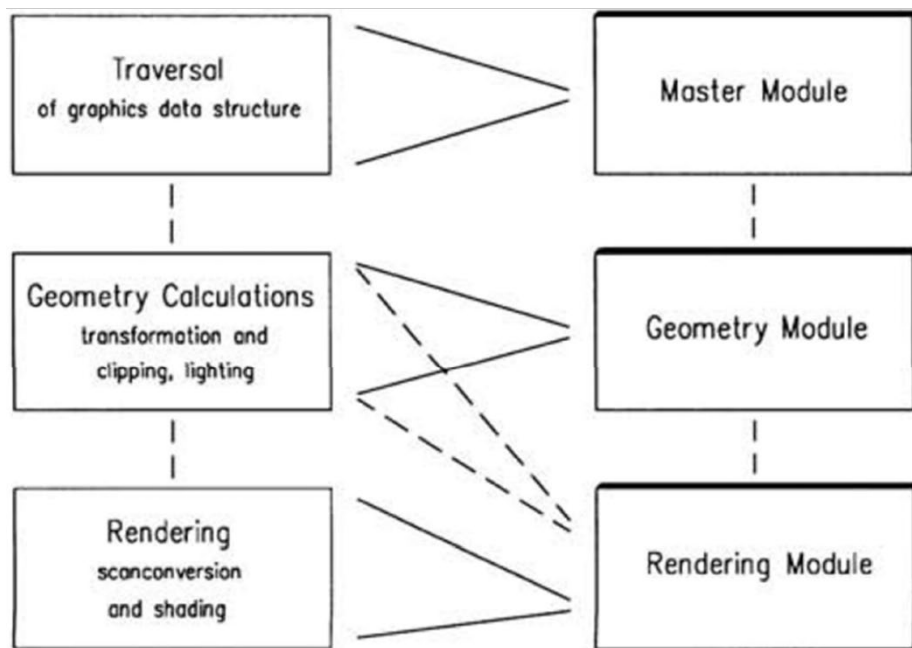


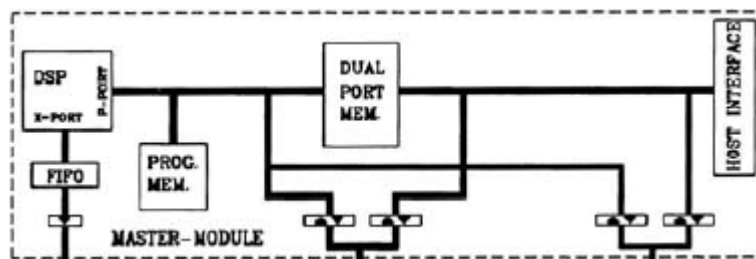
FIGURE 4.1. Mapping functional sections to hardware

Id. at 41. The operations the rendering module in Selzer may perform include generating the claimed vertex position and appearance data, which Selzer describes as transformation, clipping, polygon and patch subdivision, normal interpolation, and lighting operations. *Id.* at 42; EX1003 ¶234. In addition, the rendering module generates a pixel color and conveys the calculated pixel colors and values to the framebuffer. *Id.* The vertex and pixel operations are performed based upon the respective vertex and pixel data and instructions that are selected from the registers under coordination of the master module. *Id.*

Accordingly, Selzer, either alone or in combination with Fiske meets claim elements 2[b] and [d].

(d) Claim 2[c] - Sequencer

Selzer, either alone or in combination with Fiske, meets claim 2[c]. EX1003 ¶¶236-39. For example, the master module in Selzer accesses the graphics data in the system and “fetches the next primitive or task automatically.” EX1009 at 43.



Id. at 44. Selzer describes the master module as the “systems supervisor” that “is responsible for start-up and synchronizing activities.” *Id.* at 40. The master module gathers the graphics data and tasks and supplies those tasks (data and instructions) to the appropriate processor unit. *Id.* at 41. As noted above, the master module is able to switch between sending vertex or pixel tasks to each rendering unit, allowing them to load balance among vertex and pixel graphics operations. Once calculated, the data is stored in the local memories on each unit and eventually sent to the framebuffer. *Id.*; EX1003 ¶236.

Although Selzer’s system discloses the claimed sequencer, when combined with the scheduling module and multiple thread contexts of Fiske, this combination discloses claim element 2[c] in even more detail and in the context of a multithreaded processing unit. EX1003 ¶237. That is, when the multiple thread contexts of Fiske are present in a rendering module of Selzer, the Fiske scheduling algorithm would choose the active thread for which the rendering module would execute the next instruction. *Id.* That instruction for the particular thread would either be for a vertex calculation or for a pixel calculation. EX1003 ¶238. Because Fiske assigns a priority to each and every thread in the system, the scheduling and instruction selection

may be very fine-grained and lead to an efficient and dynamic system that is able to move back and forth between vertex and pixel operations very rapidly. *Id.*

Accordingly, the combination of Selzer and Fiske renders claim 2 unpatentable as obvious.

5. Independent Claim 3

(a) Claim 3 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 3[a] and [c] – Processor Unit

As discussed above for claim 2, Selzer’s system includes a rendering module that prioritizes pixel calculations. EX1009 at 46-47. Selzer notes that its system is dynamic and can adjust to process either vertex or pixel operations in the rendering module. EX1003 ¶241. In one example, Selzer describes how the rendering module prioritizes pixel operations, but if it reaches an idle state (*i.e.*, no pixel resources are available), then it switches to prioritize and perform vertex operations. *Id.*; EX1009 at 47. Selzer notes that while pixel processing performance is reduced in this situation, the gains in switching to prioritizing and performing vertex operations outweigh that loss and provide for a more efficient processing unit. *Id.* A POSITA would have understood that Selzer’s “automatic load balancing” architecture

prioritizes either vertex or pixel operations depending on the “processing load” present in the system. EX1003 ¶241.

Even though Selzer fully discloses all elements of claim 3[a] and 3[c], when combined with Fiske, the scheduling of vertex or pixel operations becomes even more fine-grained and dynamic. That is, and as explained above for claim 2, the scheduling algorithm in Fiske with its multiple thread contexts can load balance in two ways. First, if enough shared resources are not available for pixel processing, then the Selzer/Fiske combination would allocate its thread contexts for vertex processing. EX1010 at 210-211; EX1003 ¶242. The same would hold true if enough shared resources were not available for vertex processing, then Fiske would prioritize and process pixel operations. *Id.* Second, once those thread contexts are allocated, Fiske performs another selection to determine which resources should be used for vertex or pixel operations. *Id.* Fiske describes this process being done in a fine-grained manner that can switch among different instructions of a thread, and at a time when the entire thread is not completed. *Id.* This further meets the limitations of claim 11, because the multithreaded processor would perform vertex or pixel operations based on vertex or pixel workload, and can switch between instructions in the instruction store and perform vertex and pixel operations at various degrees of completion (*e.g.*,

switch between unfinished vertex threads and unfinished pixel threads).

EX1003 ¶242. In this manner, the combination of Selzer and Fiske would work in a multithreaded environment and meet claim 3, as well as the workload-related limitations of claims 1, 4, 5 and 11, directed to the processor unit and sequencer/instruction store.

(c) Claim 3[b] – Shared Resources

Petitioner incorporates by reference the discussion *supra* at §2[a].

6. Independent Claim 4

(a) Claim 4 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 4[a] and [c] – Processor Unit

See supra at §3[a] and [c].

(c) Claim 4[b] – Shared Resources

See supra at §3[b].

7. Independent Claim 5

(a) Claim 5 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 5[a] – Processor Unit

See supra at §2[b].

(c) Claim 5[b] – Sequencer

See supra at §§2[c], 3[a] and [c]. EX1003 ¶248 (discussing the

scheduling architecture of Fiske being able to dynamically adjust to select any type of data available based on resources).

8. Dependent Claim 6

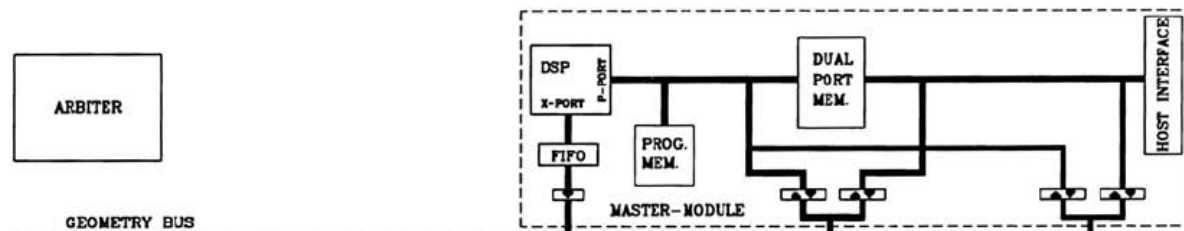
Dependent claim 6 recites, “The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.” As explained *supra* at §2[c], Selzer’s master module has direct access to the busses and fetches the appropriate data to supply to each rendering module. This meets claim 6.

9. Dependent Claims 7 and 10

Dependent claim 7 recites, “The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.” Dependent claim 10 recites, “The shader of claim 7, wherein the control signal is provided by an arbiter.”

As explained *supra* at claim 3[a] and [c], Selzer teaches that its master module makes the decisions of what types of operations a rendering module will perform. In doing so, the master module includes a selection circuit (shown below) that provides information to the memories in the rendering module in response to a control signal generated by the DSP in the master module. EX1003 ¶251. The selection process in the arbiter of the DSP determines whether the rendering module will perform vertex or pixel

operations. *Id.* Because the DSP in the master module is making a choice among several options, it is performing the function of an arbiter. *Id.*



The combination of Selzer and Fiske also teaches that a circuit (*e.g.*, master module, scheduling arbiter in Fiske) that provides information (*i.e.*, data) to the store (*i.e.*, memories and FIFOs) in response to a signal dictating whether to process vertex or pixel data. Because the modules in Selzer and Fiske are deciding what data to select based on the variable status of the system (*e.g.*, the switching between vertex and pixel processing in the rendering modules in Selzer, or the context switching disclosed in Fiske), this process is performed by an arbiter (in Selzer as noted above). EX1003 ¶252. In this combination, the hardware scheduler in Fiske “chooses the loaded thread with the highest priority as the next thread using simple hardware, in order to minimize context switch overhead.” EX1010 at 211. In this case, the “hardware scheduler” is the claimed arbiter, and the “simple hardware” within the arbiter provides the claimed control signal. EX1003 at 252 (also noting software implementation that meets the claims).

The threads in Fiske are vertex or pixel threads that fetch data from a cache, the claimed “store.” EX1003 at 252. “If a thread is waiting for a memory reference to be satisfied then it is stalled and is not considered for scheduling until the memory reference is satisfied.” EX1010 at 211.

“[E]very single memory reference is simulated through the cache [5, 9] since the hit rates for instructions and 'local data are typically very high.” *Id.* at 212. Accordingly, Fiske, in combination with Selzer, teaches dependent claim elements 7 and 10.

10. Dependent Claim 8

Dependent claim 8 recites, “The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected data.” This functionality is discussed *supra* at §2[d].

11. Dependent Claim 9

Dependent claim 9 recites, “The shader of claim 5, wherein the processor unit generates vertex position and appearance data in response to a selected data.” This functionality is discussed *supra* at §2[d].

12. Independent Claim 11

(a) Claim 11 [preamble] – Unified Shader

See supra at §2[preamble].

(b) Claim 11[a] and [c] – Processor Unit

See supra at §§2[c], [d], 3[a], [c]. EX1003 ¶256 (explaining how Fiske’s scheduler can switch between threads at various levels of completion in order to dynamically load balance).

(c) Claim 11[b] – Instruction Store

The “instruction store” is met by the same components that satisfy the “sequencer” limitation of claim 2. *See supra* at §§2[c], [d], 3[a], [c]. Selzer specifically refers to instruction storage and notes that the processors each include up to 256k*32 bit wide “fast static memory for instruction and data storage.” EX1009 at 40.

13. Independent Claim 1

See supra at Claims 2 and 3. Claim 1 is a method claim that puts the limitations of apparatus claim 3 into method claim format. For example, the steps to be performed in claim 1 align with the functional elements of the processor unit, and Selzer can dynamically prioritize vertex operations based on workload and space available in the general-purpose register block. EX1003 ¶258. The “instruction store” is met for the same reasons as the “sequencer” of claim 2.

Accordingly, claim 1 is met and rendered obvious for the same reasons as claims 2 and 3.

VIII. CONCLUSION

Petitioner requests that the Board grant this petition for *inter partes* review.

Dated: January 2, 2024

PERKINS COIE LLP
11452 El Camino Real, Ste 300
San Diego, California 92130-2080
858.720.5700 (phone)
858.720.5799 (fax)

Respectfully submitted,

/John Schnurer/

Lead Counsel
John Schnurer, Reg. No. 52,196

Back-up Counsel
Yun (Louise) Lu, Reg. No. 72,766
Kevin J. Patariu, Reg. No. 63,210
Kyle R. Canavera, Reg. No. 72,167

Attorneys for PETITIONER

CERTIFICATE OF WORD COUNT UNDER 37 CFR §42.24(d)

Under 37 C.F.R. §42.24(d), the undersigned certifies that the word count for this *Petition for Inter Partes Review* totals 13,997, excluding the parts exempted by 37 C.F.R. §42.24(a).

The word count was made using the built-in word count function in the Microsoft® Word software used to prepare this document.

Dated: January 2, 2024

Respectfully submitted,

PERKINS COIE LLP
11452 El Camino Real, Ste 300
San Diego, California 92130-2080
858.720.5700 (phone)
858.720.5799 (fax)

/John Schnurer/
Lead Counsel
John Schnurer, Reg. No. 52,196

Back-up Counsel
Yun (Louise) Lu, Reg. No. 72,766
Kevin J. Patariu, Reg. No. 63,210
Kyle R. Canavera, Reg. No. 72,167

Attorneys for PETITIONER

CERTIFICATE OF SERVICE

The undersigned hereby certifies that true copies of this PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 8,760,454 and supporting materials (Exhibits 1001-1016 and Power of Attorney) have been served by FedEx® mail delivery service in their entirety on the date indicated below on Patent Owner at the correspondence address of record at the U.S. Patent and Trademark Office as follows:

Volpe Koenig - AMD
30 South 17th Street -18th Floor
Philadelphia, PA 191035

Courtesy copies of this Petition and accompanying exhibits were also served via electronic mail on Patent Owner's counsel of record in a related district court litigation:

Adam S. Rizk
MINTZ, LEVIN, COHN, FERRIS, GLOSKY, AND POPEO PC
One Financial Center Boston, MA 02111
Email: AMD-ITC@mintz.com

Dated: January 2, 2024

Respectfully submitted,

PERKINS COIE LLP
11452 El Camino Real, Ste 300
San Diego, California 92130-2080
858.720.5700 (phone)
858.720.5799 (fax)

/John Schnurer/
Lead Counsel
John Schnurer, Reg. No. 52,196

Back-up Counsel
Yun (Louise) Lu, Reg. No. 72,766
Kevin J. Patariu, Reg. No. 63,210
Kyle R. Canavera, Reg. No. 72,167

Attorneys for PETITIONER