UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

REALTEK SEMICONDUCTOR CORP.,
Petitioner,

v.

ATI TECHNOLOGIES ULC
Patent Owner.

Case No. IPR2023-00922
U.S. Patent No. 8,760,454

DECLARATION OF DR. WILLIAM MANGIONE-SMITH REGARDING U.S. PATENT NO. 8,760,454

TABLE OF CONTENTS

Page

I. INTRODUCTION 1

II. SUMMARY OF CONCLUSIONS 2

A. Summary of Conclusions Regarding Reduction to Practice, Conception, and Diligence 2

B. Summary of Conclusions Regarding Validity 3

III. EXPERT QUALIFICATIONS 3

IV. HIGH LEVEL GRAPHICS PROCESSING OVERVIEW 7

A. Graphics Processors Use Vertex and Pixel Data to Create Display Images 7

B. Graphics Processors Use Instruction "Threads" To Transform Vertex and Pixel Data 10

C. Conventional Graphics Processors Executed Vertex And Pixel Threads Using Separate Vertex Shaders And Pixel Shaders 11

V. OVERVIEW OF THE '454 PATENT 14

A. The '454 Patent's Unified Shader 15

B. The '454 Patent's Unified Shader Determines Which Data to Process by Evaluating Storage Capacity and Additional Analyses As Seen In Claims 1, 3-10 16

C. The Unified Shader Can Simultaneously Execute Vertex and Pixel Operations and Switch Quickly Between Operations at Various Degrees of Completion As In Claim 11 18

D. The Invention of the '454 Patent Triggers Execution by Transmitting Data Rather Than Instructions As In Claim 2 19

E. The '454 Patent's Challenged Claims 20

VI. RELEVANT LEGAL STANDARDS 20

A. Conception and Reduction to Practice 21

B. Claim Construction 22

C. Obviousness 23

VII. GENERAL DESIGN AND DEVELOPMENT PROCESS AT AMD 24

VIII. CONCEPTION 28

A. '454 Patent 30

1. Claim 1 30

2. Claim 2 41
3. Claim 3 58
4. Claim 4 65
5. Claim 5 73
6. Claim 6 83
7. Claim 7 85
8. Claim 8 90
9. Claim 9 93
10. Claim 10 97
11. Claim 11 100

IX. DILIGENCE IN REDUCING TO PRACTICE 107

X. CONSTRUCTIVE REDUCTION TO PRACTICE 151

A. '454 Patent 153
1. Claim 1 153
2. Claim 2 159
3. Claim 3 170
4. Claim 4 180
5. Claim 5 190
6. Claim 6 195
7. Claim 7 197
8. Claim 8 198
9. Claim 9 199
10. Claim 10 200
11. Claim 11 201

XI. NO ABANDONMENT, SUPPRESSION, OR CONCEALMENT OF R400 207

XII. ASSERTED PRIOR ART 208

A. Lindholm '685 209
B. Amanatides 211
C. Selzer 212

XIII. EVALUATION OF PETITIONER'S PROPOSED GROUNDS 214

A. Ground 1: Lindholm '685 and Lindholm '913 Do Not Render the '454 Patent Obvious 214
1. No Executing Operations Depending Upon an Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4) 214

2. No Execution of Instructions "In Response to" Receiving Selected Data (Claim 2) 219
3. No "Memory" Separate From the "Store" (Claim 6) 221
4. No "Control Signal" (Claim 7) or "Arbiter" (Claim 10) 222
5. No Processor Unit That Performs Vertex Manipulation Operations and Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11) 223
B. Ground 2: Amanatides and Kohn Do Not Render the '454 Patent Obvious 226
1. No Executing Operations Depending Upon An Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4) 226
2. No Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11) 234
3. No "Selected Data" (Claims 2, 5) 237
4. Alleged "Selection Circuit" Is Not Within Alleged "Unified Shader" As Required (Claims 7, 10) 239
5. No "Control Signal" (Claims 7, 10) 242
6. No "Selection Circuit" or "Arbiter," "(Claim 10) 243
C. Ground 3: Selzer and Fiske Do Not Render the '454 Patent Obvious 245
1. No Executing Operations Depending Upon An Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4) 246
2. No Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11) 251
3. No Execution of Instructions "In Response to" Receiving Selected Data (Claim 2) 254
4. Alleged "Sequencer," "Instruction Store," "Circuitry," "Selection Circuit," and "Arbiter" Are Not Within Alleged "Unified Shader" As Required (Claims 2, 5-7, 10, 11) 256
5. Alleged "Sequencer" And "Instruction Store" Do Not Maintain Instructions (Claims 2, 5, and 11) 258

6. No "Arbiter" (Claim 10) or "Control Signal" (Claims 7, 10) 260
7. No Motivation to Combine Selzer and Fiske and no Reasonable Expectation of Success in Doing So 261

XIV. SECONDARY CONSIDERATIONS SUPPORT NON-OBVIOUSNESS OF THE '454 PATENT 265

A. Initial Skepticism of ATI's Unified Shader Technology 267
B. Unexpected Results in Developing the Unified Shader Technology 272
C. The Satisfaction of a Long-Felt Need by the Unified Shader of the Xenos GPU and Failed Attempts by Others 274
D. Industry Praise of the Unified Shader in Xbox 360's Xenos GPU 276
E. Commercial Success of the Xbox 360 Containing ATI's Unified Shader in the Xenos GPU 278
F. Adoption by Others 280

XV. CONCLUSION 281

TABLE OF ABBREVIATIONS

Abbreviation	Full Description
AMD	Advanced Micro Devices, Inc.
ATI	ATI Technologies, ULC
GPU	Graphics processing unit

ATI Ex. No 2121
IPR2023-00922
Page 6 of 324

TABLE OF EXHIBITS

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, *A Simple, Flexible, Parallel Graphics Architecture,* 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, *Introducing the Intel i860 64-bit Microprocessor,* IEEE, Volume 9, Issue 4, pages 15-30, August 1989 ("Kohn")
1009	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) published in 1993 ("Selzer") [Library of Congress]
1010	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) published in 1995 ("Fiske")
1016	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) published in 1993 ("Selzer") [University of California, Berkeley, Library]
EX 2001	Declaration of William Mangione-Smith and CV
EX 2002	IPR2015-00325, Declaration of Dr Wolfe, Sept. 9, 2015
EX 2003	IPR2015-00325, Declaration of Calvin Watson, Sept. 9, 2015
EX 2004	IPR2015-00325, Declaration of Lefebvre, Sept. 9, 2015

Exhibit #	Reference Name
EX 2005	IPR2015-00325,-00326, and-00330, Deposition Transcript of Calvin Watson, November 4, 2015
EX 2006	IPR2015-00325, -00326, and -00330, Deposition Transcript of Dr. Wolfe, Nov. 10, 2015
EX 2007	IPR2015-00325,-00326, and -00330, Deposition Transcript of Laurent Lefebvre, Nov.13, 2015
EX 2008	Gruber et al, R400 Shader Processor 2001, Oct. 9, 2015
EX 2009	R400 Document Library Folder History, November 1, 2000 through April 8, 2005
EX 2010	R400 Sequencer Specification (Version 0 4), August 14, 2001
EX 2011	R400 Sequencer Specification (Version 2 0), September 24, 2001
EX 2012	R400 Shader Processor (Version 0 1), Jan. 23, 2001
EX 2013	R400 Top Level Specification (Version 0 2), March 11, 2001
EX 2014	R400 Program Logs
EX 2015	Microsoft Site Visit (Feb. 2003)
EX 2016	Log of Exhibits and File Location
EX 2017	R400 Program Review Documentation (Feb. 2003)
EX 2018	R400 MM Software Status (Feb. 2003)
EX 2019	R400 I/O Presentation (Feb. 2003)
EX 2020	R400 I/O Presentation (Feb. 2003)
EX 2021	R400 August Program Review
EX 2022	Executive Review - R400 (Oct. 2002)
EX 2023	R400 Area Estimate
EX 2024	R400 Executive Review (Sept. 2002)
EX 2025	GFIXIP 9x SX Micro-Architecture Specification
EX 2026	WD/IA VGT Micro-Architecture Specification
EX 2027	GX9 SPI Specification
EX 2028	R400 Top Level Specification (Version 0.2)
EX 2029	R400 Folder History Log
EX 2030	R400 Folder History Log
EX 2031	R400 Folder History Log
EX 2032	R400 Review PowerPoint

ATI Ex. No 2121
IPR2023-00922
Page 8 of 324

Exhibit #	Reference Name
EX 2033	R400 Review PowerPoint
EX 2034	R400 Development and Documentation
EX 2035	Xenos GFX Change History July 2003 - Dec. 2003 Log
EX 2036	R400 GFX Change History March 2003 - Dec. 2003
EX 2037	R400 Primitive Assembly
EX 2038	R400 Primitive Assembly
EX 2039	Development Documentation
EX 2040	Development Documentation
EX 2041	Development Documentation
EX 2042	R400 Performance Verification
EX 2043	Development Documentation
EX 2044	R400 PAD Program Review (Dec. 2002)
EX 2045	R400 EMU Test Regress History Log
EX 2046	R400 EMU Test Regress Statistics Log
EX 2047	R400 Program Review (Dec. 2002)
EX 2048	R400 Program Review (Dec. 2002)
EX 2049	Device Development Progress
EX 2050	Device Development Progress
EX 2051	Device Development Progress
EX 2052	Device Development Progress
EX 2053	R400 Review Status PowerPoint
EX 2054	TV Specification
EX 2055	R400 Program Review PowerPoint and Test Results
EX 2056	Development Documentation
EX 2057	R400 PowerPoint
EX 2058	R400 Program Review PowerPoint
EX 2059	R400 Program Review
EX 2060	Block Development Progress
EX 2061	R400 Program Review (Dec. 11, 2002)
EX 2062	R400 Program Review (Dec. 12, 2002)
EX 2063	Development Documentation
EX 2064	Development Documentation

ATI Ex. No 2121
IPR2023-00922
Page 9 of 324

Exhibit #	Reference Name
EX 2065	Development Documentation
EX 2066	R400 Technical Documentation
EX 2067	R400 Technical Documentation
EX 2068	R400 GFX Change History March 2003 to December 2003 Log
EX 2069	Ikos 2002 Spreadsheet
EX 2070	R400 - Program Review (Oct. 2002)
EX 2071	Sqsp Regress Report
EX 2072	Xenos Sq Change Log
EX 2073	R400 Regress Testing
EX 2074	Virtual Logic 3.1 User's Guide
EX 2075	Virtual Logic 3.5.5 User's Guide
EX 2076	IKOS Virtual Logic 2.1
EX 2077	IKON Screenshots
EX 2078	IKON Screenshots
EX 2079	GFX9 User's Guide
EX 2080	US Patent No. 6,897,871
EX 2081	Samsung Exynos 5430 Octa SoC
EX 2082	Samsung Exynos 3 Quad 3470 Processor Database
EX 2083	HW Emulator First Triangle
EX 2084	HW Simulator First Triangle
EX 2085	R400 Regress Testing Logs
EX 2086	R400 Regress Testing Logs
EX 2087	R400 IKOS Status
EX 2088	R400 Program Review
EX 2089	R400 Program Review Documentation
EX 2090	PA Check-in History Log
EX 2091	RB Check-in History Log
EX 2092	SC Check-in History Log
EX 2093	SPI Check-in History Log
EX 2094	SP Check-in History Log
EX 2095	SQ Check-in History Log
EX 2096	SX Check-in History Log

Exhibit #	Reference Name
EX 2097	VGT Check-in History Log
EX 2098	R400 File Logs 10/1/2002 – 4/17/2003
EX 2099	Netlist Area Sheets
EX 2100	IKOS Schematic
EX 2101	Stephen Morein Depo Transcript (AMD v. LG) May 25, 2017
EX 2102	Laurent Lefebvre Depo Transcript (337-TA-1044) June 28, 2017
EX 2103	IPR2015-00325, Declaration of Laurent Lefebvre
EX 2104	Development Documentation
EX 2105	Development Documentation
EX 2106	Development Documentation
EX 2107	Development Documentation
EX 2108	Development Documentation
EX 2109	Development Documentation
EX 2110	IPR2015-00325, EX 2018 - EX. 2056
EX 2111	IPR2015-00325, EX 2057 - EX 2071
EX 2112	Block Change Logs
EX 2113	Andrew Gruber Depo Transcript (AMD v. LG) July 27, 2017
EX 2114	IPR2015-00326, EX 2001 - EX2002
EX 2115	IPR2015-00326, EX 2003
EX 2116	IPR2015-00326, EX 2004
EX 2117	IPR2015-00326, EX 2005
EX 2118	IPR2015-00326, EX 2006
EX 2119	IPR2015-00326, EX 2007 - EX 2072
EX 2120	IPR2015-00326, EX 2073 - EX 2118
EX 2121	Supplemental Declaration of William Mangione-Smith
EX 2122	Supplemental Declaration of Calvin Watson
EX 2123	Deposition Transcript of Hanspeter Pfister (02/16/2024)
EX 2124	Beyond3D-ATI Xenos: Xbox 360 Graphics Demystified – July 26, 2017
EX 2125	Dictionary of Computing 5th Ed (excerpt)
EX 2126	Microsoft Computer Dictionary 5th Edition (excerpt)
EX 2127	How GPUs Work, David Luebke, Greg Humphreys

Exhibit #	Reference Name
EX 2128	2006 Microsoft Corporation Annual Report
EX 2129	ATI and NVIDIA Proclaim Different Graphics Processors Architecture Goals, Anton Shilov (December 23, 2004)
EX 2130	NVIDIA Chief Architect: Unified Pixel and Vertex Pipelines – The Way to Go. NVIDIA Says It Would Make a Chip with Unified Pipes "When it Makes Sense" Anton Shilov (July 11, 2005)
EX 2131	FORTUNE: Review of the Xbox 360 – Nov. 17, 2005
EX 2132	TECHSPOT – History of the Modern Graphics Processor, Part 3 – The Fall of 3Dfx and The Rise of Two Giants, Graham Singer (April 10, 2013)
EX 2133	Working of Xbox 360 – How Xbox 360 Works (February 11, 2010)
EX 2134	Xbox 360: the Ars Technica Review, Ben Kuchera (November 30, 2005)
EX 2135	Microsoft Xbox dominates a sluggish 2011 gaming market, Nathan Pensky (January 13, 2012)
EX 2136	Microsoft's Xbox 360, Sony's PS3 – A Hardware Discussion, Anand Lal Shimpi & Derek Wilson (June 24, 2005)
EX 2137	Qualcomm History and its GPU (R)evolution – Ryan Shrout (June 22, 2015)
EX 2138	Xbox 360 vs. PlayStation 3: The Hardware Throwdown – IGN, Jesse Schedeen (August 26, 2010)
EX 2139	Xbox 360 and Playstation 3's Graphics Card (GPU) Compared – Red Gaming Tech, CrimsonRayne (March 11, 2013)
EX 2140	Xbox 360 GPU – NeutralX2 (April 26, 2006)
EX 2141	Wikipedia - Adreno
EX 2142	Game Consoles: Global Market share 2006-2010 – Statista
EX 2143	NVIDIA – Frequently Asked Questions
EX 2144	The Xbox 360 Uncloaked – The Real Story Behind Microsoft's Next-Generation Video Game Console, Dean Takahashi
EX 2145	Xbox 360 System Architecture
EX 2146	Interview with Ken Kutaragi, President of Sony – "Why I worked with NVIDIA on the PS3" (Certified Translation)

Exhibit #	Reference Name
EX 2147	Scott Wasson, "Details of ATI's Xbox 360 GPU unveiled," The Tech Report (May 19, 2005)

ATI Ex. No 2121
IPR2023-00922
Page 13 of 324

I. INTRODUCTION

1. I have been retained by ATI Technologies UCL ("ATI" or "Patent Owner"), as an independent expert in this proceeding before the Patent Trial and Appeal Board ("PTAB" or "Board"), to examine whether United States Patent No. 8,760,454 ("the '454 Patent") was conceived and diligently reduced to practice prior to the Lindholm '685 and '913 patents, whose effective dates are June 30, 2003 and June 27, 2003, respectively. I understand that Realtek requested the Board institute an Inter Partes Review ("IPR") proceeding of the '454 Patent (EX. 1001) in IPR2023-00922, and that the Board instituted the IPR on December 1, 2023.

2. I am providing this testimony for consideration by the Patent Trial and Appeal Board in this proceeding. I have personal knowledge of the facts and opinions set forth in this declaration and all the opinions and conclusions found in this declaration are my own.

3. I am being compensated at my ordinary hourly rate for my efforts with regards to this matter. My compensation does not depend in any way on the outcome of this proceeding or the opinions I express, or the testimony I give.

4. This declaration contains my conclusions and a summary of my analysis including a summary of my conclusions, an overview of my qualifications as an expert, an overview of the materials I have considered in arriving at my conclusions, an overview of the terminology and legal principles that I applied in

my analysis, an overview of the technical background of the subject matter, an overview of the ’454 Patent, and an analysis of when the inventions of the ’454 Patent were conceived and diligently reduced to practice.

5. This declaration is based on information currently available to me. I intend to continue my investigation and study, which may include a review of documents and information that may yet be produced, as well as deposition testimony from depositions for which transcripts are not yet available or that may yet be taken in this proceeding. Therefore, I expressly reserve the right to expand or modify my opinions as my investigation and study continue, and to supplement my opinions in response to any additional information that becomes available to me, any matters raised by Petitioner (or its experts) or the Board (or its experts), or in light of any relevant opinions or orders from the Patent Trial and Appeal Board or other authoritative body. Moreover, I reserve the right to provide rebuttal testimony regarding any analyses and opinions raised in opposition to my declaration.

II. SUMMARY OF CONCLUSIONS

A. Summary of Conclusions Regarding Reduction to Practice, Conception, and Diligence

6. It is my opinion that the historical ATI design documentation demonstrates that at least by at least August 24, 2001, the inventors had conceived of the inventions claimed in claims 1-11 of the ’454 Patent. It is also my opinion that the inventors diligently reduced the inventions of the ’454 Patent to practice. It is

my opinion that the filing of US Patent Application 10/718,318 application, a parent application to the application which resulted in the '454 Patent, on November 20, 2003 qualifies as a constructive reduction to practice of claims 1-11 of the '454 Patent. US Patent Application 10/718,318 resulted in U.S. Patent No. 6,897,871 (the "'871 Patent"). The specification of the '871 Patent discloses each and every element of claims 1-11 of the '454 Patent.

B. Summary of Conclusions Regarding Validity

7. It is my opinion that the '454 Patent is not obvious in view of: (i) Lindholm '685 and Lindholm '913; (ii) Amanatides and Kohn; and (iii) Selzer and Fiske. It is also my opinion that secondary considerations supports validity of the claims of the '454 Patent.

III. EXPERT QUALIFICATIONS

8. My technical background and experience cover most aspects of computer system design, including low level circuitry, computer architecture, computer networking, graphics, application software, client-server application, Web technology, and system software (*e.g.*, operating systems and compilers). I am a member of the Institute of Electrical and Electronics Engineers and the Association for Computing Machinery, which are the two most significant professional organizations in my profession. I have been employed as a design engineer, research engineer, professor, and technical expert. Over my professional career, I have been

an active inventor with 112 issued U.S. patents, 200 published and pending U.S. patent applications and many unpublished U.S. patent applications.

9. From 1984 until 1991, I attended the University of Michigan in Ann Arbor, Michigan, where I was awarded the degrees of Bachelor of Science and Engineering, Master of Science and Engineering, and Doctorate of Philosophy. My doctoral research focused on high performance computing systems including computer architecture, applications and operating system software, and compiler technology. One of my responsibilities during my graduate studies included teaching senior undergraduate students who were about to enter the profession.

10. After graduating from the University of Michigan, I was employed by Motorola in Schaumburg, Illinois. While at Motorola, I was part of a team designing and manufacturing the first commercial battery-powered product capable of delivering Internet email over a wireless (i.e., radio frequency) link and one of the first personal digital assistants. I also served as the lead architect on the second-generation of this device with control over the entire system design including the memory subsystem architecture, embedded processor, ASIC, power system, and analog circuitry. Part of my responsibilities at Motorola involved the specification, design, and testing of system control Application-Specific Integrated Circuits ("ASICs"). I conducted the initial research and advanced design that resulted in the Motorola M*Core embedded microprocessor. M*Core was designed to provide the

high performance of desktop microprocessors with the low power of contemporaneous embedded processors. The M*Core received widespread use in many communications products including various cellular handsets, advanced pagers, and embedded infrastructure. While at Motorola I was the sole inventor on one U.S. patent.

11. From 1995 until 2005, I was employed by the University of California at Los Angeles ("UCLA") as a professor of Electrical Engineering. I was the director of the laboratory for Compiler and Architecture Research in Embedded Systems ("CARES") and served as the field chair for Embedded Computing Systems. The CARES research team focused on research, engineering, and design challenges in the context of battery-powered and multi-media mobile computing devices. One of the key developments of my lab was the Mediabench software tool, which is widely used to design and evaluate multi-media embedded devices. Key elements of Mediabench include software that is essential for modern digital wireless communications. My primary responsibility, in addition to classroom teaching, involved directing the research and training of graduate students. I was a tenured member of the faculty and had responsibilities for teaching as well as scholarly research. While at UCLA I was a named inventor on three U.S. patent applications, one of which issued as a patent. My colleagues at UCLA were some of the leading scientists and engineers in the world with a long list of innovations from computer

network security devices to the nicotine patch. The graduate student researchers in my laboratory came from a diverse set of backgrounds, all with undergraduate degrees in computer engineering, electrical engineering, or computer science, many with multiple years of experience working as professional engineers in areas such as software development, computer system design and ASIC circuit design.

12. From 2005 until 2009, I was employed at Intellectual Ventures in Bellevue, Washington. My responsibilities at Intellectual Ventures included business development, technology assessment, market forecasting, university outreach, collaborative inventing, intellectual property licensing support, and intellectual property asset pricing. My colleagues and co-inventors at Intellectual Ventures included the former lead intellectual property strategist at Intel, Intel's former lead IP council, Microsoft's former chief software architect, the founder of Microsoft research, the designer of the Mach operating system, the architect of the U.S. Defense Department's Strategic Defense Initiative, the founder of Thinking Machines (a seminal parallel processing computer system), and Bill Gates. I had responsibility for hiring and managing over 15 staff members including multiple Ph.Ds. with degrees in electrical engineering and decades of experience in product design and engineering.

13. A summary of some of my qualifications for forming the opinions in this declaration are as follows: I have more than 30 years of experience as a computer

architect, computer system designer, educator, and as an executive in the PC and electronics business. I am also a member of several professional associations, such as the ACM, IEEE and have been intimately involved in professional research through the International Symposium on Microarchitecture (Program Chair for 26th and General Chair for 36th), IEEE Transactions on Computers (Associate Editor), ACM Transactions on Embedded Computing Systems (Associate Editor), and IEEE Computer (Associate Editor). I also have been on the program committees for ISCA, MICRO, ISLPED, Network Processors Workshop, FPL, Complexity-Effective Design, RAW, Workshop on Mediaprocessors, and DSP, FPT, and INTERACT.

14. For further details regarding my employment and academic history, please refer to my *curriculum vitae* Appendix A.

IV. HIGH LEVEL GRAPHICS PROCESSING OVERVIEW

15. The ’454 Patent discloses novel architectures for graphics processing unit circuitry. Consumer products are often used to generate and display graphics on an output device such as a built-in screen or an external monitor. When complex and realistic graphics are desired, there is often additional specialized circuitry, in the form of a chip, which is added to the consumer product to assist it with the complex processing that it must perform to render the graphics to the screen. This specialized circuitry is known as a graphics processing unit or “GPU.”

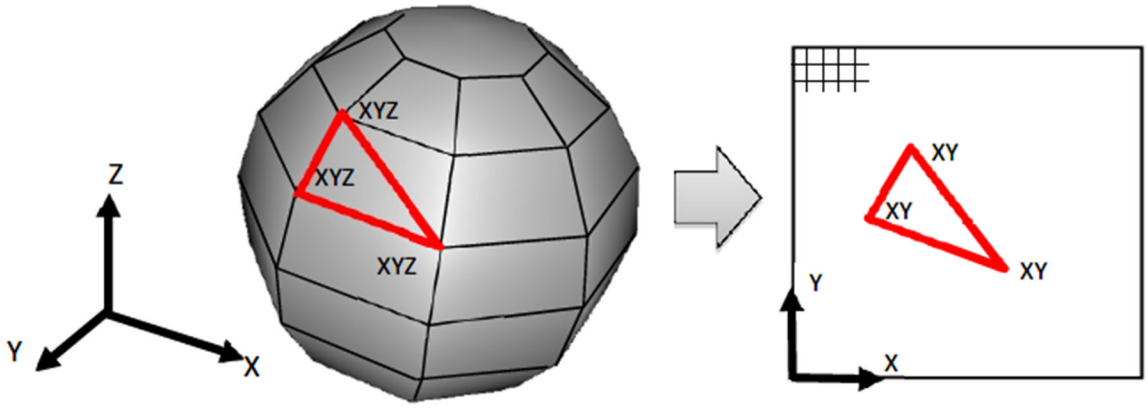
A. Graphics Processors Use Vertex and Pixel Data to Create Display Images

16. Graphics processors (GPUs) are designed to convert a three-dimensional object into an image for display on a two-dimensional screen. As part of this process, a three-dimensional object is rendered as a collection of simple shapes, called primitives. For example, in the figure below, a three-dimensional character has been rendered as a collection of triangular primitives.

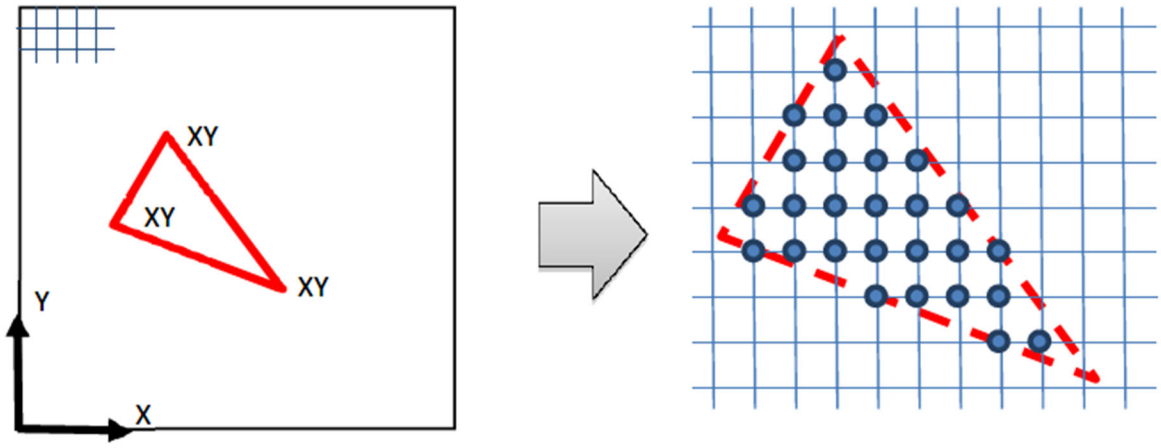


17. Each triangular primitive is defined by the positions of its three corner points, i.e., its "vertices." After a three-dimensional object is rendered as a group of primitives, subsequent processing steps include converting the vertices of each primitive from three-dimensional coordinates to two-dimensional coordinates. To illustrate, in the example shown below, a triangular primitive is being used to render a portion of a sphere and has vertices with three-dimensional position coordinates

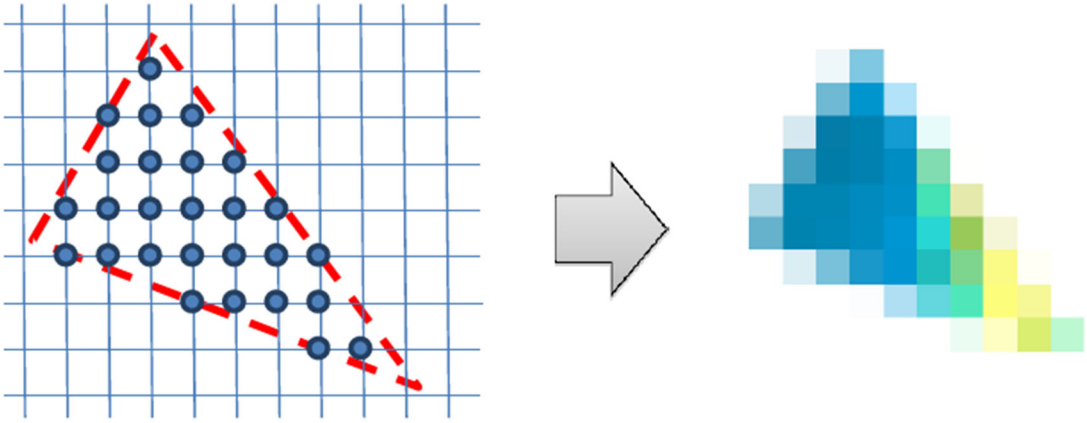
along the X-, Y-, and Z- axes. These three-dimensional vertices coordinates are converted into two-dimensional coordinates (along only the X- and Y- axes) for display on a two-dimensional screen.



18. Subsequent processing steps include rendering each primitive as two-dimensional collection of dots on a screen called "pixels." As part of this process, the graphics processor uses the two-dimensional vertices coordinates to determine which pixels fill a particular primitive. In the illustration below, the pixels that fill the primitive defined by the XY coordinates are depicted in blue.

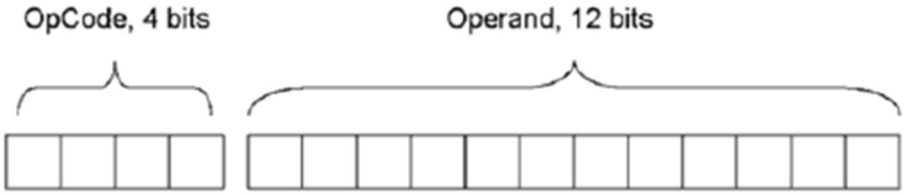


19. Further processing includes identifying the correct color and texture for each pixel filling the primitive.



B. Graphics Processors Use Instruction "Threads" To Transform Vertex and Pixel Data

20. Graphics processors use various collections of instructions to transform a three-dimensional object into a two-dimensional display. Each collection of instructions is called a "thread." Each thread contains a sequence of instructions to be performed one after another. Each instruction consists of an "OpCode" and an "Operand." The OpCode indicates which calculation operation is required by the instruction, such as an arithmetic or logical operation. The Operand indicates the location of the graphics data upon which the calculation operation is to be performed.



An Exemplary 16-bit Instruction

21. Some threads comprise instructions that manipulate and transform vertex data. These threads are known as "vertex threads," and can include instructions that, for example, work together to transform a triangular primitive's vertex coordinates from three-dimensional to two-dimensional (as discussed above), to translate or scale the primitive, to remove any portion of the triangular primitive that may be off the screen, or to cull primitives that do not project onto the screen.

22. Other threads comprise instructions that manipulate and transform pixel data. These threads are known as "pixel threads," and can include instructions that, for example, work together to transform pixel data by assigning color value to the pixels, determining lighting effects, determining texture coordinate values, and blending colors.

C. Conventional Graphics Processors Executed Vertex And Pixel Threads Using Separate Vertex Shaders And Pixel Shaders

23. Conventional graphics processors featured several circuits, each configured to perform a single type of transformation. These circuits are known as "shaders."

24. Conventionally, each shader was dedicated to executing a particular type of thread to transform a particular type of data. For example, a "vertex shader" would execute only vertex threads to transform vertex data, such as, for example, transforming the three-dimensional coordinates of a primitive's vertices in object space into two-dimensional coordinates for a display screen. EX. 1001 at 1:63-2:7. But to execute pixel threads to transform pixel data, another, separate shader was needed. This type of shader is known as a "pixel shader," and was used, for example, determine and apply the color for each pixel making up a rendered object. *Id.* at 2:8-18.

25. Thus, to execute the position and color threads and operations necessary to transform a three-dimensional object to a two-dimensional display, a conventional graphics processor needed at least two shaders, one a vertex shader and the other a pixel shader. *Id.* at 2:15-18. As described in the '454 patent, this requirement resulted in drawbacks in both size and performance. First, the need for multiple specialized shaders increased the size of graphics processors:

> Conventional graphics processors require the use of both a vertex shader and a pixel shader in order to generate an object. Because both types of shaders are required, known graphics processors are relatively large in size, with most of the real estate being taken up by the vertex and pixel shaders.

Id. at 2:20-29.

26. Second, sequencing the pixel shader after the vertex shader meant that

one shader was dependent on – and had to wait for – the output of the other before it could continue processing, leading to idle downtime that slowed down the graphics processor and resulted in relatively poor performance:

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

Id. at 2:20-29.

27. Such a prior art "pipeline" is illustrated in Figure 3 of the '454 patent, which depicts a vertex shader 46 in series with pixel shader 54.

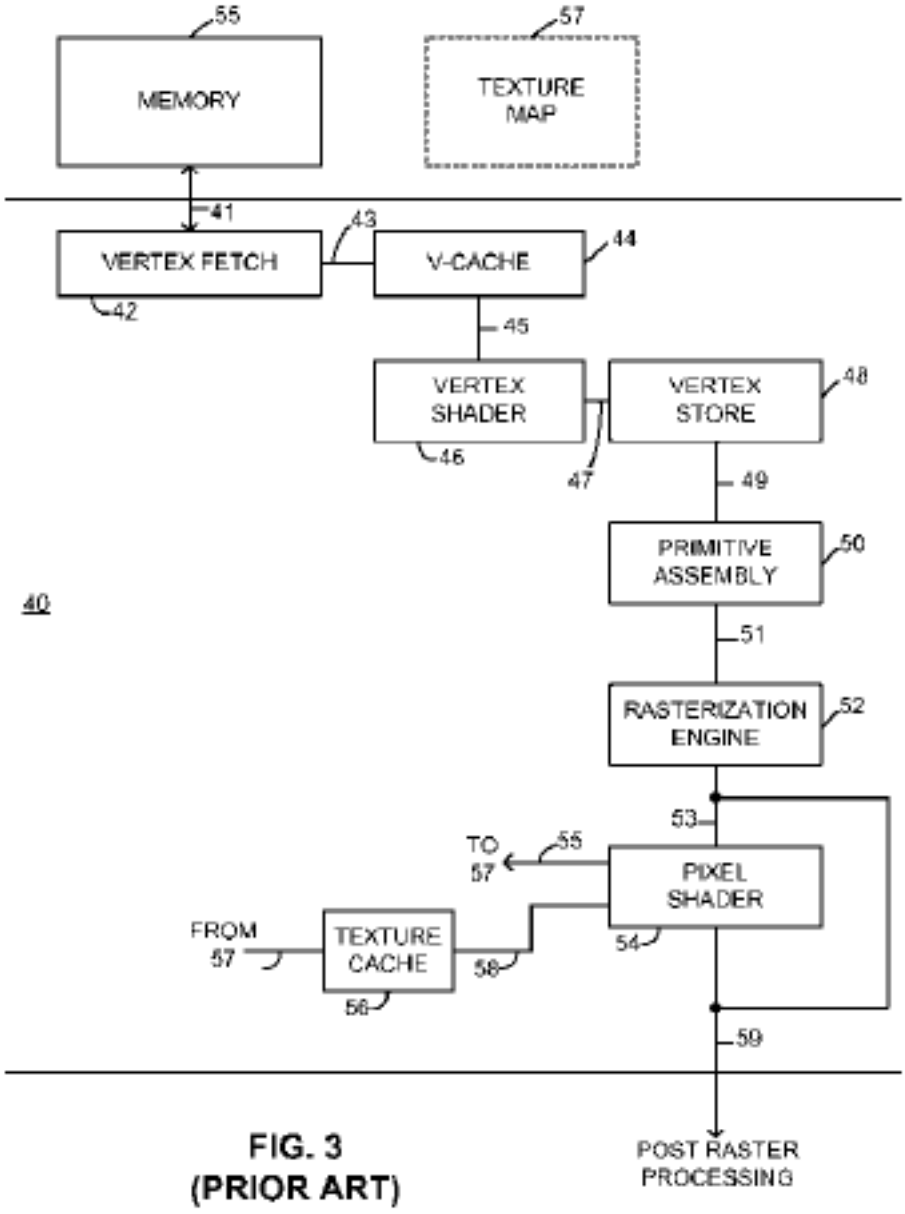


FIG. 3
(PRIOR ART)

Id. at Fig. 3

V. OVERVIEW OF THE '454 PATENT

28. The inventions of the '454 Patent are directed to "video graphics processing." Before the '454 Patent, the industry often employed general-purpose processors to render 3-D computer graphics in software. However, general purpose

processors handled graphics processing operations in an inefficient way, causing poor performance.

A. The ’454 Patent’s Unified Shader

29. As its title suggests, the ’454 patent solves the real estate and performance problems of separate vertex and pixel shaders by providing a shader that can execute both vertex threads and pixel threads to transform both vertex and pixel data, respectively. EX. 1001 at 2:58-61. This multipurpose shader “is referred to as a ‘unified’ shader.” In an embodiment described in the patent, the unified shader includes a register, a sequencer, and a processor. *Id.* at 3:4-10. The unified shader’s register stores the inputs transmitted to the unified shader, including the vertex and pixel data that are to be transformed by the unified shader. *Id.* at 3:4-5, 4:29-39.

30. The unified shader’s sequencer stores the individual instructions that, when sequenced together, comprise the vertex and pixel threads that the unified shader will use to manipulate and transform the stored vertex and pixel data. *Id.* at 3:4-10, 4:52-5:31. The ’454 patent refers to those individual instructions (*i.e.*, the logical and arithmetic operations) as vertex and pixel “calculation operations.” *Id.* at 3:4-7, 3:63-66, 4:42-45, 4:53-60, 5:36-52. The resulting threads are called vertex and pixel “manipulation operations.” *Id.* at 3:4-7. 5:19-36. The unified shader’s processor can execute the vertex and pixel instructions and threads (stored in the

sequencer) on the vertex and pixel data (stored in the register). *Id.* at 3:4-12. The unified shader is thus configured to perform as both a complete pixel shader and a complete vertex shader in one. *Id.* at 2:58-61, 3:10-12, 5:19-31.

B. The '454 Patent's Unified Shader Determines Which Data to Process by Evaluating Storage Capacity and Additional Analyses As Seen In Claims 1, 3-10

31. In contrast to prior approaches (including that of the prior art asserted here), which may use pre-determined priorities or balance workloads based on the amount of work in the queue for execution, the system of the '454 Patent describes determining whether the unified shader will process a vertex or pixel operation based on available storage space.

32. For example, the patent describes an "arbiter circuit for selecting one of a plurality of inputs for processing [by the unified shader] . . . wherein the shader performs one of the vertex operations or pixel operations based on the [arbiter's] selected one of the plurality of inputs." *Id*., 2:58-3:17. The arbiter's selection criteria, or "arbitration scheme," includes at least the capacity to store particular types of data. To this end, the data storage (the unified shader's register block) is monitored for available space, which informs what data is selected to be transmitted to the processor to be processed: "if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not

satisfied." *Id*., 5:32-44.

33. In use, for example, when there is not enough room in the data storage for vertex data, the processor "continues pixel calculation operations that are to be or are currently being performed [by] the processor based on instructions maintained in an instruction store *until enough registers within the general purpose register block become available*" to store vertex data. *Id*., Abstract (emphasis added).

34. The claims require this functionality in a variety of different ways, with different levels of specificity. Claim 1 is the most specific, requiring performing vertex operations until there is no more room in the general purpose register block for more incoming vertex data, in which case continuing to perform pixel operations until enough space appears to once again store vertex data. This requires two evaluations: first that there is no longer room for additional vertex data in the general purpose register block (at a time when the system is performing vertex operations), and second, that at a later point in time (when the system is performing pixel operations), there is once again room for additional vertex data in the general purpose register block.

35. Claims 3 and 4 are slightly broader. Claim 3 requires a processor unit to be "operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations." Claim 4 requires the opposite type of processing; the

processor unit must be "operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations."

36. Claim 5 is the broadest, requiring that the processor unit "execute vertex calculation and pixel calculation operations on selected data maintained in a store depending upon an amount of space available in the store."

C. The Unified Shader Can Simultaneously Execute Vertex and Pixel Operations and Switch Quickly Between Operations at Various Degrees of Completion As In Claim 11

37. In addition to describing a unified shader that can handle both vertex data and pixel data, the '454 Patent further describes ways to best implement such a unified shader. For example, the patent describes how to allocate the unified shader's resources among vertex and pixel data in a manner that prevents backlogs of data waiting to be processed and idle downtime for the shader. *Id.* Under the described approach, the disclosed unified shader switches back and forth between unfinished operations to prevent downtime and backlogs of data sitting waiting to be processed. *See, e.g.*, EX. 1001, 5:23-36.

38. To illustrate, when executing a vertex operation, the unified shader may encounter a vertex instruction that it cannot yet perform because the necessary data or memory is not yet available. EX. 1001, 5:36-44. Rather than having the unified shader sitting idle, waiting for the necessary data or memory so that it can proceed

with the rest of the vertex operation, the unified shader switches to pending pixel operations and executes pixel instructions until the idle vertex operation is ready to proceed. EX. 1001, 5:32-52.

39. Thus, the '454 Patent discloses a unified shader that "has the ability to simultaneously perform vertex manipulation operations [vertex operations] and pixel manipulation operations [pixel operations] ***at various degrees of completion*** by being able to freely switch between such program or instructions very quickly." *Id*., 5:32-36 (emphasis added). "[T]hrough the sharing of resources within the unified shader 62, processing of image data is enhanced as there is no down time associated with the processor 96." EX. 1001, 5:49-52. The '454 Patent thus discloses and claims a unified shader that can switch back and forth between executing ***unfinished*** vertex and pixel operations. *Id.*

40. Further, claim 11 in particular requires the unified shader "perform[] the vertex manipulation operations and pixel manipulation operations *at various degrees of completion* based on *switching* between instructions in the instruction store." EX. 1001, claim 11 (emphasis added).

D. The Invention of the '454 Patent Triggers Execution by Transmitting Data Rather Than Instructions As In Claim 2

41. The '454 patent specifically provides the receipt of selected data is what causes the processor to perform the corresponding operations: "the shader performs one of the vertex operations or pixel operations *based on the selected one of the*

plurality of inputs." *Id*., 2:58-3:17. This approach is in contrast with the approaches in the asserted references, in which the shader performs tasks *based on the received instructions, **not** the received data as claimed*. This data-driven approach where the receipt of particular selected data dictates the execution of instructions enables the invention's ability to quickly switch between partially-completed tasks, as the data drives the decision-making about what to process, rather than the instructions doing so.

42. This approach is seen in claim 2, which requires, in relevant part: "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."

E. The '454 Patent's Challenged Claims

43. The Petition challenges claims 1-11 of the '454 patent. Challenged claims 1, 2, 3, 4, 5, and 11 are independent claims. Challenged claims 6-10 depend directly or indirectly from independent claim 5.

VI. RELEVANT LEGAL STANDARDS

44. In preparing and expressing my opinions and considering the subject matter of the '454 Patent, I am relying on certain basic legal principles that counsel have explained to me. These principles are discussed below.

A. Conception and Reduction to Practice

45. Under pre-AIA 35 U.S.C. § 102(e), "[a] person shall be entitled to a patent unless . . . the invention was described in . . . a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent." 35 U.S.C. § 102(e).

46. I understand that a patent owner can swear behind a prior art reference by antedating the reference, *i.e.*, by establishing that the reference's priority date occurred *after* the patent owner's invention. To do so, a patent owner must show: **(1)** prior conception, **and** **(2)** (a) prior reduction to practice or (b) reasonably continuous diligence from the reference's priority date to the invention's reduction to practice (*i.e.*, the patent's filing date).

47. I understand that a prior reduction to practice can be either actual (i.e., an embodiment of the claimed invention) or constructive (i.e., a filed patent application).

48. I understand that to establish an actual reduction to practice, the patent owner must prove that: (1) the inventor(s) constructed an embodiment in a tangible form or performed a process that met all the limitations and (2) the inventor(s) determined that the invention would work for its intended purpose.

49. I understand that courts apply a reasonableness standard in weighing the sufficiency of evidence of reduction to practice.

50. I understand that for diligence, the inventor(s) need not prove that they continuously exercised reasonable diligence through the critical period, where the inventor worked on reducing the invention to practice every day during the critical period. Rather, the inventor(s) must show they exercised reasonably continuous diligence, and that periods of inactivity do not automatically extinguish a claim of reasonable diligence.

51. I understand that to satisfy conditions of patentability, the inventor(s) may not abandon, suppress, or conceal the invention.

52. When interpreting the claim terms and assessing reduction to practice, I have attempted to interpret the claims of each of the Asserted Patents from the perspective of a person of ordinary skill in the relevant art, in the context of the patent documents.

B. Claim Construction

53. I understand that the first step in determining the validity of a claim is for the claim to be properly construed. I have been further advised that, in an *inter partes* review proceeding, the claims of a patent are typically given their ordinary and customary meaning as would be understood by one of ordinary skill in the art in the context of the claims and the patent disclosure (specification).

54. I further understand that an inventor can provide special definitions for specific claim term(s) within the patent. However, any special definition for a claim

term must be set forth in the specification with reasonable clarity and precision.

C. Obviousness

55. I have been informed that a claim may be invalid as obvious if the subject matter described by the claim as a whole would have been obvious to a POSITA at the time the claimed invention was made. I understand that the standard for obviousness in an *inter partes* review proceeding is by a preponderance of the evidence.

56. I have also been informed that a determination of obviousness involves an analysis of the scope and content of the prior art, the similarities between the claimed invention and the prior art, and the level of ordinary skill in the art. I have been informed and understand that a prior art reference should be viewed as a whole.

57. I have been informed that in considering whether an invention for a claimed combination would have been obvious, I may assess whether there are apparent reasons to combine known elements in the prior art in the manner claimed in view of interrelated teachings of multiple prior art references, the effects of demands known to the design community or present in the market place, and/or the background knowledge possessed by a POSITA. I also understand that other principles may be relied on in evaluating whether a claimed invention would have been obvious.

58. I have been informed that, in making a determination as to whether or

not the claimed invention would have been obvious to a POSITA, one may consider certain objective indicators of non-obviousness if they are present, such as: commercial success of product(s) practicing the claimed invention; long-felt but unsolved need; teaching away; unexpected results; copying; and praise by others in the field. I understand that for such objective evidence to be relevant to the non-obviousness of a claim, however, there must be a causal relationship (called a "nexus") between the claim and the evidence. I also understand that this nexus must be based on a novel element of the claim rather than something available in the prior art.

59. I have also been informed and understand that when considering the obviousness of a patent claim, one should consider whether a reason or motivation existed for combining the elements of the references in the manner claimed, and that the prior art must create a reasonable expectation of success in producing the claimed subject matter.

VII. GENERAL DESIGN AND DEVELOPMENT PROCESS AT AMD

60. At a high level, once an idea has been conceived at ATI and AMD, inventors and their colleagues work diligently to implement it. Such effort entails, among other things, extensive efforts to write and debug the hardware (RTL files), test the RTL to confirm that it worked for the purpose intended by the concept, and construct physical embodiments. *See*, *e.g.*, EX. 2004 (Lefebvre Decl., IPR2015-

00325), at pp. 21-27; 33-60; *see generally* EXS. 2110-2112, EX. 2103, EXS. 2104-2109; EX. 2118; EX. 2119.

61. Work is corroborated through (1) continuous updates to technical specifications; (2) development, testing, and debugging of the RTL, including all critical components; and (3) synthesis, placement, and routing of netlists on prototype systems, for example the IKOS FPGA prototype. *See* EX. 2004 (Lefebvre Decl., IPR2015-00325), at pp. 21-27; 33-60; *see generally* EXS. 2110-2112, EX. 2103, EXS. 2104-2109; EX. 2118; EX. 2119.

62. During development, the architectural lead engineers for each block write specifications to describe the structure and function of the blocks. *See generally*, *e.g.*, EX. 2008; EX. 2011; EX. 2037; EXS. 2066-2067; EX. 2104; EXS. 2105 to 2109. Revisions of these specifications are commonly updated every two to three weeks, and multiple revisions during development is typical. *See*, *e.g.*, EX. 2104; EXS. 2105 to 2109 (outlining edits to the document). Each revision will demonstrate the progress made during the previous week(s), and during interim periods, the project team will work diligently on outstanding issues. *See id.*

63. Work performed on a project is saved in a revision control system called Perforce. *See* EX. 2003 (Watson Decl., EX. 2105 325 IPR), at pp. 10-11; *see generally* EX. 2005 (IPR2015-0325, 326, 330 Watson Dep.). Perforce is a revision-controlled repository that maintains time stamped logs, user permissions, and

whether files are checked out. *See* EX. 2003 (Watson Decl., EX. 2105 325 IPR), at pp. 10-11. As such, Perforce will capture every change made to files describing the design and development of a project at ATI. *See* EX. 2009; EXS. 2029; EX. 2030; EX. 2031; EX. 2035; EX. 2036; EX. 2068; EXS. 2110-2111; *see also* EX. 2102 (Lefebvre Dep.), at 208:12-201:13, 219:19-221:20.

64. Throughout the development process, project teams will also track their efforts in program review presentations. *See* EX. 2014; EXS. 2016-2023 EX. 2034; EXS 2039-2043; EX 2044-2055; EX 2056-2065; EX. 2070; EX. 2119. These presentations serve to catalogue the efforts of the team, and include information related to the development of all deliverables for the project, including technical specification drafting, emulation code development, RTL code development, verification and validation of all emulation and RTL features, netlist synthesis, prototype testing, hardware development, software development, physical design layout, and other topics necessary to build a product for sale. *See id.*

65. Development teams also maintain in Perforce all changes made to the emulation and RTL block level development files in the form of changelists and file logs. *See* EX. 2003 (Watson Decl., EX. 2105 325 IPR), at pp. 66; EX. 2004 at ¶ 39; EX. 2007 at 157:15-158:7; *see generally* EX. 2005 (IPR2015-0325, 326, 330 Watson Dep.). These files serve to identify the date a change was made, the individual(s) making the change, and what change was made. *Id.*

66. As features of each critical block are completed, regression testing is utilized to confirm that the iterations of the emulation and RTL code maintain the finalized features. *See* EXS. 2085-2086; EXS. 2071-2073; EX. 2007 at 125:25-126:6. Regression test files indicate the date the tests are run, the block tested, the test used, and the results of the tests. Regression testing is continually conducted to ensure future versions of developed emulation and RTL code continue to exhibit previously-developed functionality. *Id.*

67. Once feature sets are completed for critical blocks, netlists are synthesized from stabilized versions of RTL. *E.g.*, EX. 2007 at 32:8-34:8, 101:3-12, 104:22-105:5, 114:8-12, 123:24-130:2. The netlists are generated on a block level and integrated with a chip level netlist. Multiple versions of netlists are generated as the project team diligently works to reduce the concept to practice.

68. After netlists are synthesized, the project team diligently works to place and route the netlist as a constructed embodiment of the concept on a prototype system. *E.g.*, EX. 2007 at 105:6-106:25. Once placed and routed, tests are run on the prototype system, which implement the function of the block level RTL. *Id.*

69. Critical milestones during the development process may include emulation completion, Verilog feature completion, netlist synthesis, prototype implementation, and commercial product release. Additionally, a critical verification step of development may include execution of the "1st Triangle Test" in both

software emulation testing and Verilog testing of the RTL itself. *E.g.*, EX. 2007 at 115:22-118:7; EX. 2004 at ¶¶ 44-52. The 1st Triangle Test involves the rendering of a 3D triangle. Completion of this test indicates that the features of the essential blocks of a graphics processing unit operate for their intended purpose, – *i.e.*, rendering 3D graphics. *Id.* In the case of a unified shader, the 1st Triangle Test also exercises the unified shader to confirm that it can process both pixels and vertices. *Id.* By passing the 1st Triangle Test, the RTL is demonstrated to be stable and operating for its intended purpose throughout the remainder of the development cycle, including prototyping. *Id.* Additional regression testing of both emulation and RTL code is performed to verify that subsequent versions of RTL still pass the triangle test.

70. Netlist synthesis and generation is then utilized by the physical design team to generate a floorplan for the chip. EX. 2007 at 34:9-35:9, 243:6-19.

VIII. CONCEPTION

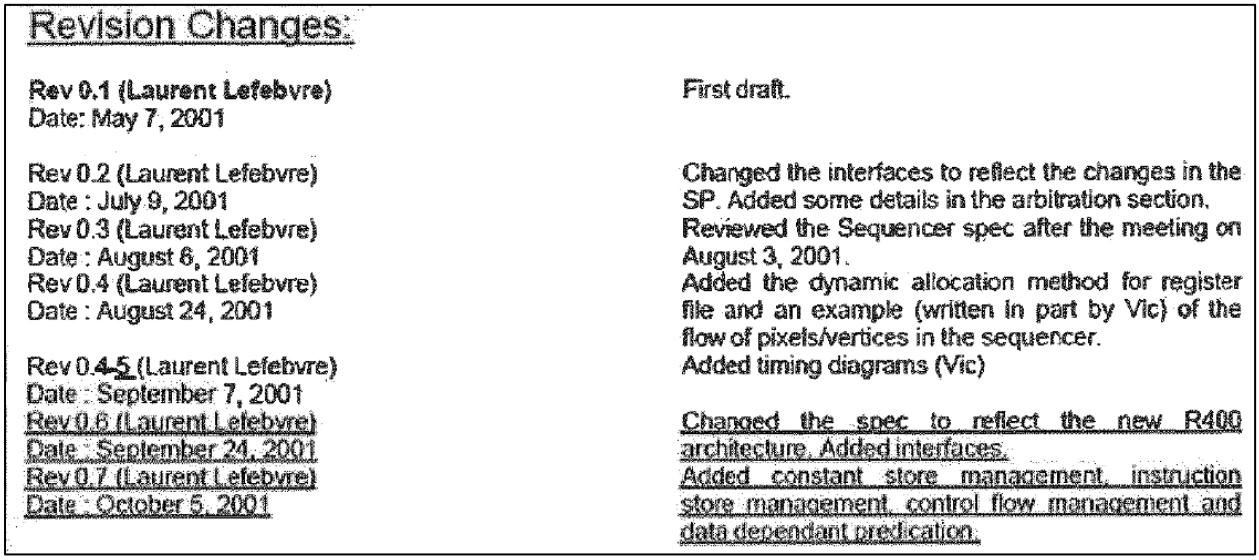
71. I understand that within US Patent law the conception of "conception" requires a mental formulation and disclosure by the inventor(s) of a complete idea for an apparatus or process. I further understand that conception relies upon the inventor(s)' ability to describe the invention with particularity, and conception must be sufficiently complete to enable a person of ordinary skill in the art to reduce the concept to practice. Based on my review of the relevant documents, listed below, it

is my opinion that the claims of the '454 Patent were conceived at least by August 24, 2001.

72. I have reviewed R400 Sequencer Specification SEQ, Version 0.4 (herein "Sequencer Specification"), the R400 Top Level Specification, Version 0.2 (herein "Top Level Specification", and the Shader Processor, Version 0.1 (herein "Shader Specification") to better understand the inventors' state of mind. EX. 2010 (R400 Sequencer Spec); EX. 2013 (R400 Top Level Spec); EX. 2012 (R400 Shader Processor). The Sequencer Specification Version 0.4 was last revised by Lefebvre on August 24, 2001. EX. 2010 (R400 Sequencer Spec), at p. 3; EX. 2117 (IPR2015-0325, 326, 330 Watson Decl.), at ¶¶ 33-35. The R400 Top Level Specification Version 0.2 was last revised by Morein on March 14, 2001. EX. 2013 (R400 Top Level Spec), at p. 4; EX. 2117 (Watson Decl., EX. 2105 326 IPR), at ¶¶ 124-126. The R400 Shader Processor Version 0.1 was last revised by Skende on May 9, 2001 or May 10, 2001. EX. 2012 (R400 Shader Processor), at p. 3; *see also* EX. 2003 (Watson Decl., EX. 2105 325 IPR), at ¶¶ 127-129.

73. As the last of the revision dates in these documents is August 24, 2001, they collectively demonstrate conception of the '454 patent no later than August 24, 2001. These dates were authenticated by Calvin Watson, an employee at ATI. *See generally* EX. 2003 (Watson Decl., EX. 2105 325 IPR); EX. 2117 (Watson Decl., EX. 2105 326 IPR). As Mr. Watson explained, the last date in the revision history

portion of the document is the last time it was actually edited. *Id.* at ¶¶ 14-15. Further, as he explained, although some of the documents have dates later than August 24, 2001 under a heading "edit date," that date does not relate to the substance of the document itself, but rather is an auto-populated macro, which cannot be disabled, that records the last time a document was merely opened. *Id*. To record actual edits to the document, it is the regular practice of ATI to record the date that a technical specification was edited for substance within the document itself, in the "revision changes" or similar section such as the example below. *Id.* at ¶ 15.



Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001 Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001 Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section. Reviewed the Sequencer spec after the meeting on August 3, 2001. Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer. Added timing diagrams (Vic)
Rev 0.~~4~~5 (Laurent Lefebvre) Date : September 7, 2001 Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001 Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces. Added constant store management, instruction store management, control flow management and data dependant predication.

74. Additionally, other R400 technical documents support the following element-by-element analysis. *See generally* EX. 2037; EXS. 2066-2067.

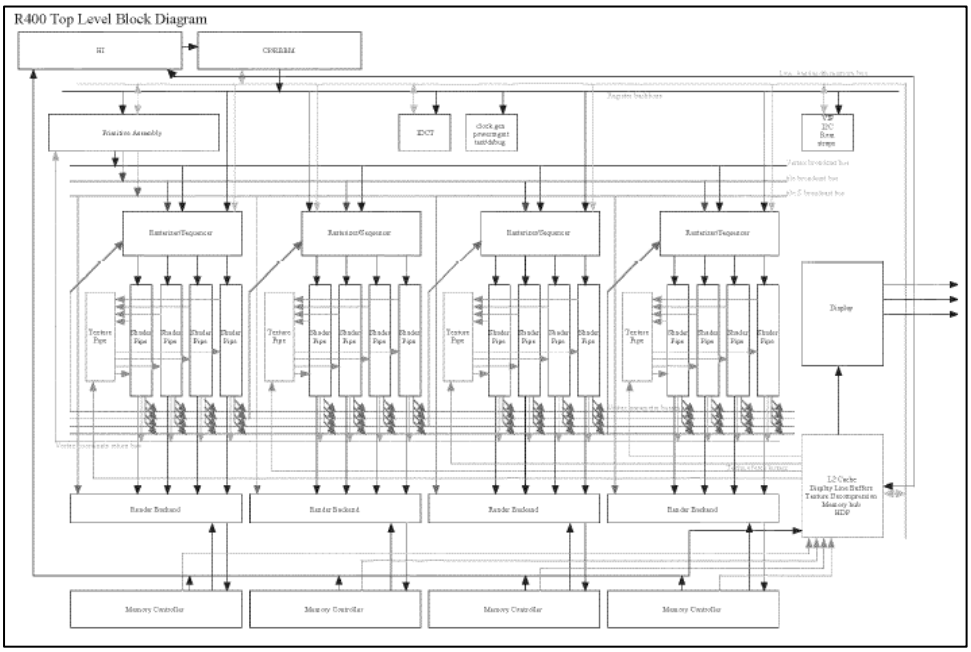
A. '454 Patent

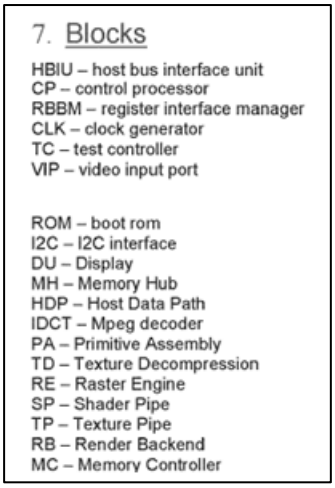
1. Claim 1

b. "A method carried out by a unified shader comprising:"

75. The preamble of Claim 1 recites "A method carried out by a unified shader comprising."

76. It is my opinion that the R400 is a unified shader which carries out the claimed method. The AMD documentation in the 2001 timeframe describes a unified shader performing the same steps as the unified shader of the '454 Patent. AMD documentation as of August 24, 2001 described this element. For example, the Top Level Specification states that the R400 is a "standalone graphics chip product." *See* EX. 2013 (R400 Top Level Spec), at p. 6. Thus, the R400 is a graphics processor. Further, the Top Level Specification describes the R400 as having the following functional blocks including a shader processor and sequencer.





7. Blocks

HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller

See EX. 2013 (R400 Top Level Spec), at pp. 15-16.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See Id. at p. 11.

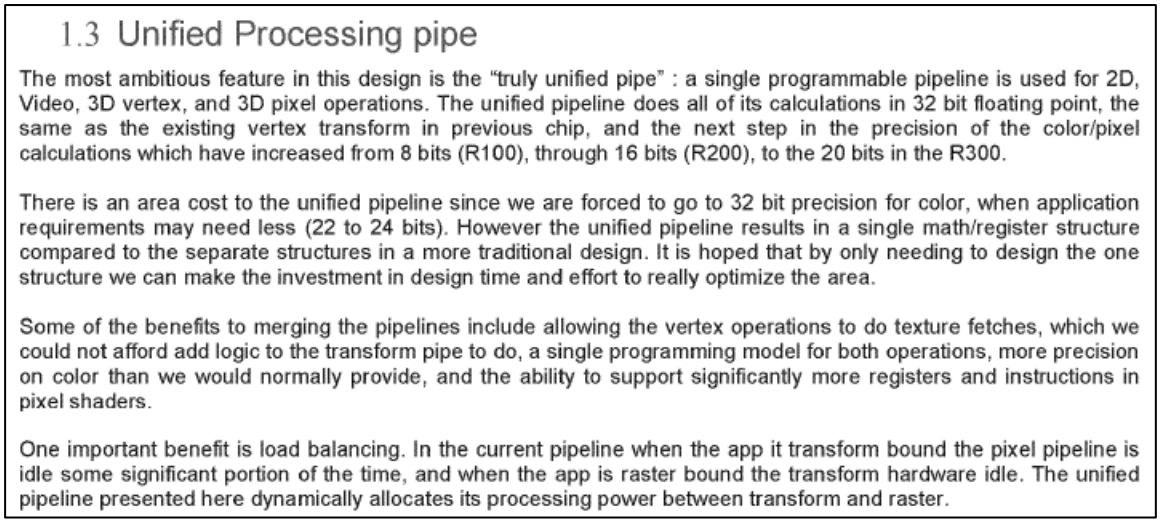
8.15 RE – Raster Engine

8.15.1 *Description*

The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

77. Moreover, the R400 implements a unified processing pipe to perform 3D vertex and pixel operations.



1.3 Unified Processing pipe

The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

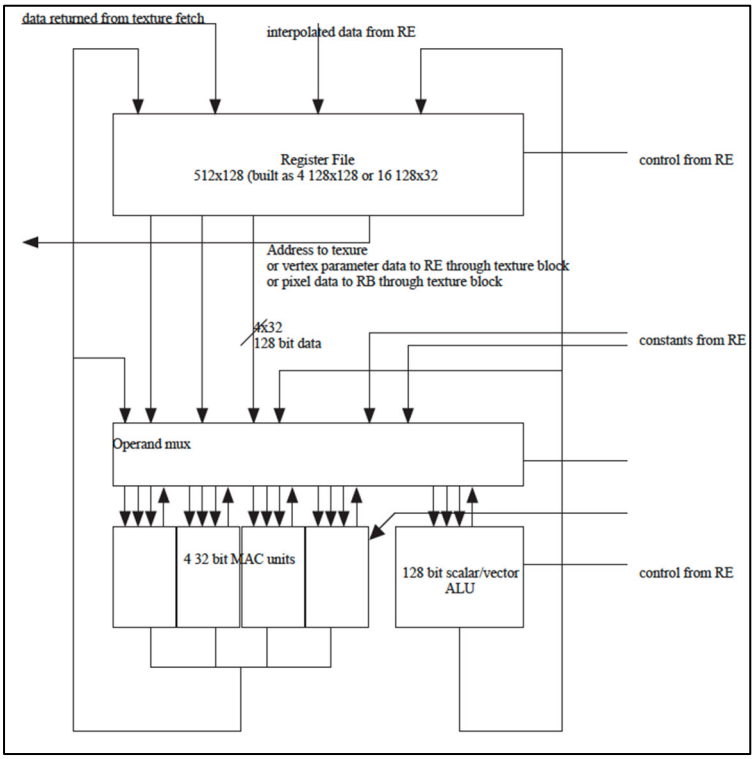
One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

78. Specifically, the R400 utilizes a unified shader, which comprises a SIMD/vector engine that performs pixel shader operations and vertex shader operations, to render 3D graphics. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-12. The unified processing pipeline of the R400, including its unified shader, performs 3D rendering by transforming vertices and determining the color of pixels. *See* EX. 2013 (R400 Top Level Spec), at pp. 10-12.

79. The passages above describe the unified shader as disclosed in the patent specification. The unified processing pipeline is a single pipeline that performs both pixel and vertex operations, achieve computation efficiency via load balancing among the different types of work, and space savings via area optimization. As described by Mr. Gruber and the above documentation, the unified shader is described as a vertex shader when it is performing vertex operations, and a pixel shader when performing pixel operations. *See* EX. 2113 (Gruber Dep.) at

100:19-101:1. Additionally, the following graphical depiction demonstrates the structure of the unified processing pipeline.



See EX. 2013 (R400 Top Level Spec), at p. 31.

The above disclosure also illustrates that the unified shader as conceived can be a single graphics processor component. *See Id*.

c. "performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and"

80. The first element of Claim 1 recites "performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and."

81. AMD documentation as of August 24, 2001, described a unified shader

performing the same steps as the unified shader of the '454 Patent. For example, the unified shader implements dynamic load balancing between transform and raster.

> One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

82. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel manipulation operations until enough registers within the general purpose register become available, at which point the unified shader will perform vertex manipulation operations.

> 2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
> - at this point the vector is removed from the Vertex FIFO
> - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 17.

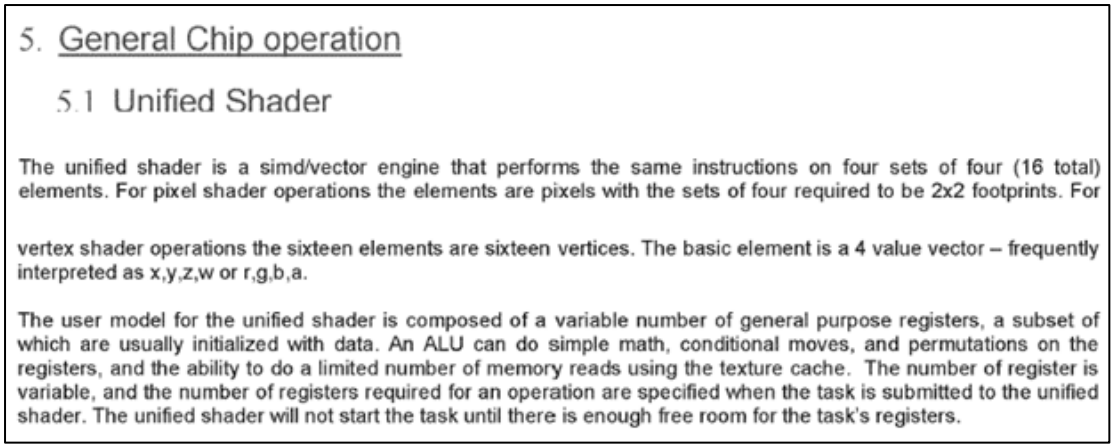
> 4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 18.

> The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

83. Additionally, the ALUs of the unified shader initialize the general purpose register block with either vertex data or pixel data.



5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

d. "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"

84. The second element of Claim 1 recites "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."

85. AMD documentation as of August 24, 2001, described a unified shader performing the same steps as the unified shader of the '454 Patent. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel calculation operations until enough space on the general purpose register block becomes available for the storage of vertex data, at which point the unified shader will perform vertex calculation operations.

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 17.

4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 18.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

e. "continuing pixel calculation operations that are to be or are currently being performed by the processor"

86. The third element of Claim 1 recites "continuing pixel calculation operations that are to be or are currently being performed by the processor."

87. AMD documentation as of August 24, 2001, described a unified shader performing the same steps as the unified shader of the '454 Patent. The processor unit of the unified shader performs vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the store. For example, the unified shader implements dynamic load balancing between transform and raster.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See EX. 2013 (R400 Top Level Spec), at p. 7.

88. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel calculation operations until enough space on the general purpose register block becomes available for the storage of vertex data, at which point the unified shader will perform vertex calculation operations.

> 2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
> - at this point the vector is removed from the Vertex FIFO
> - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 17.

> 4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 18.

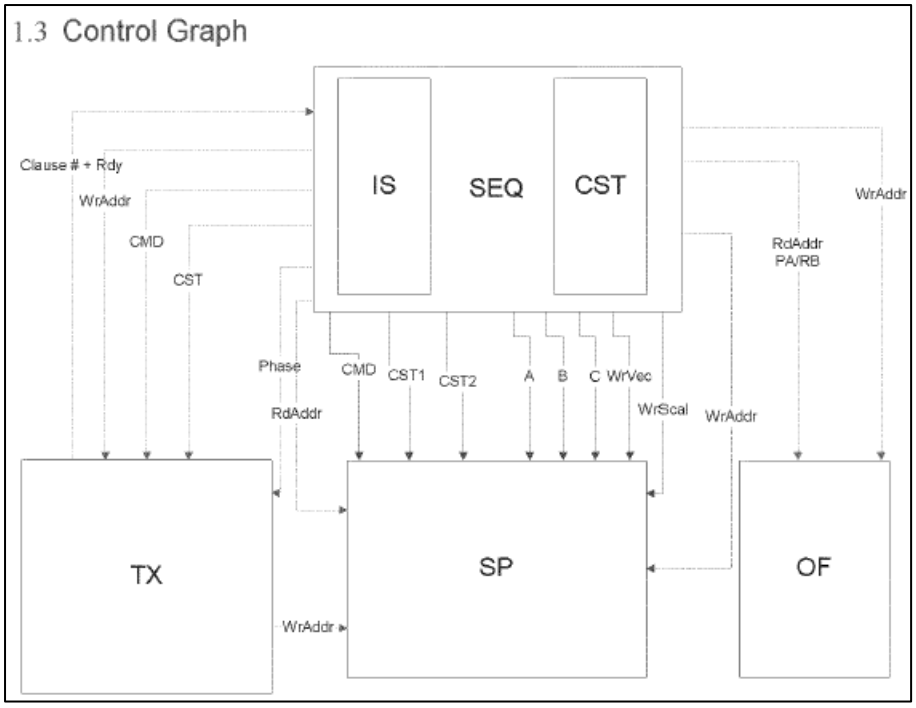
> The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

f. "based on instructions maintained in an instruction store until enough registers within the general purpose register block become available."

89. The fourth element of Claim 1 recites "based on instructions maintained in an instruction store until enough registers within the general purpose register block become available."

90. AMD documentation as of August 24, 2001, described a unified shader performing the same steps as the unified shader of the '454 Patent. For example, as stated previously, the R400 includes an instruction store.



See Id. at p. 12.

91. The processor unit of the unified shader performs vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the store. For example, the unified shader implements dynamic load balancing between transform and raster.

> One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See EX. 2013 (R400 Top Level Spec), at p. 7.

92. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel manipulation operations until enough registers within the general purpose register become available, at which point the unified shader will perform vertex manipulation operations.

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 17.

4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 18.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

93. As a result, the ALU clauses are executed in an interleaved manner so as to hide the ALU latency.

The sequencer is based on the R300 design. It chooses an two ALU clauses and a texture clause to execute, and executes all of the instructions in aa clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the top bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

See Id. at p. 4.

94. Accordingly, there will always be two active ALU clauses at any given time.

There will always be two active ALU clauses at any given time (and two arbitrers) In this case, the instructions of a vector are interleaved with the instructions of the other vector. One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

See Id. at p. 6.

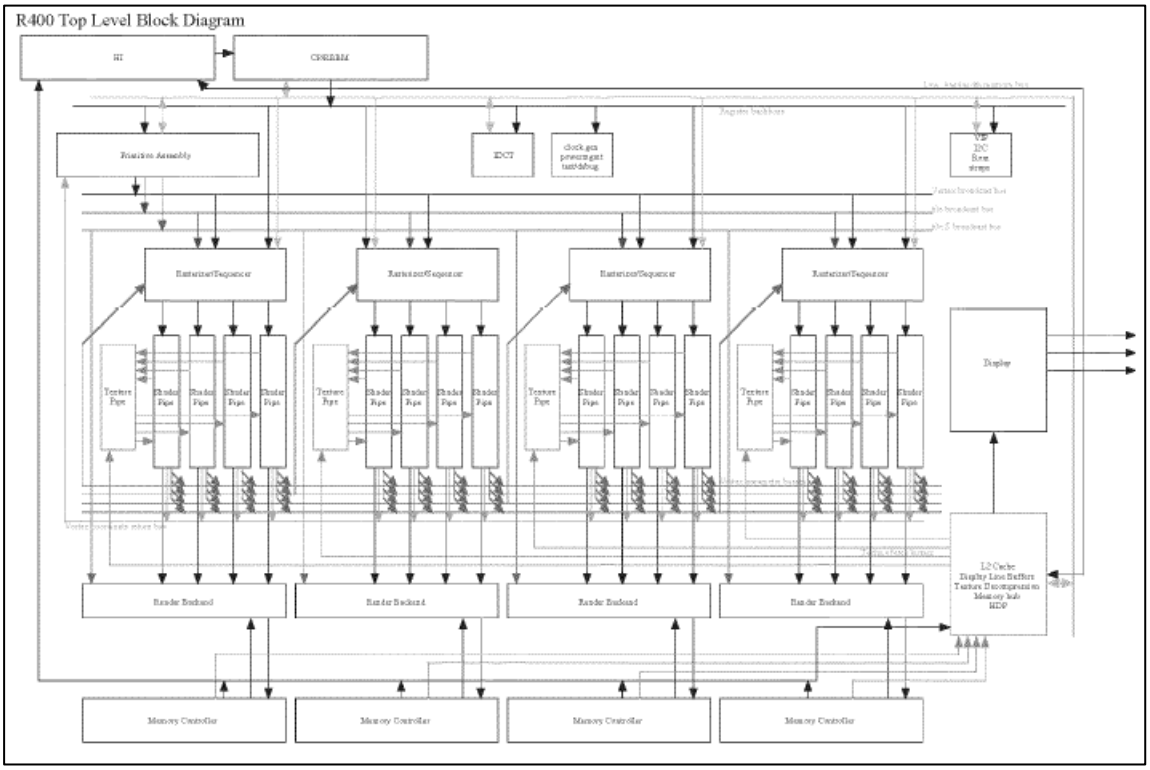
2. Claim 2

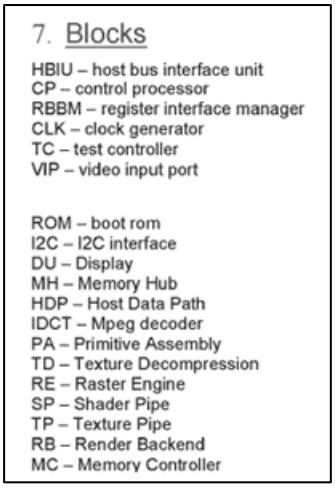
a. "A unified shader, comprising"

95. The preamble of Claim 2 recites "A unified shader, comprising."

96. It is my opinion that the R400 is a unified shader.

97. The AMD documentation in the 2001 timeframe describes a unified shader having the same characteristics as the unified shader of the '454 Patent. AMD documentation as of August 24, 2001 described this element. For example, the Top Level Specification states that the R400 is a "standalone graphics chip product." *See* EX. 2013 (R400 Top Level Spec), at p. 6. Thus, the R400 is a graphics processor. Further, the Top Level Specification describes the R400 as having the following functional blocks including a shader processor and sequencer.





7. Blocks

HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller

See EX. 2013 (R400 Top Level Spec), at pp. 15-16.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See Id. at p. 11.

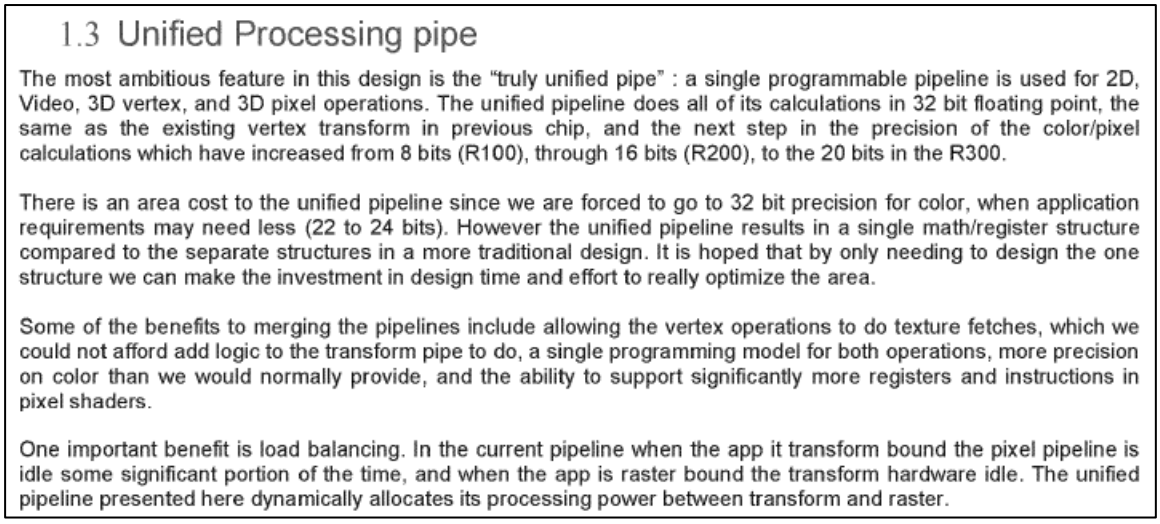
8.15 RE – Raster Engine

8.15.1 *Description*

The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

98. Moreover, the R400 implements a unified processing pipe to perform 3D vertex and pixel operations.



1.3 Unified Processing pipe

The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

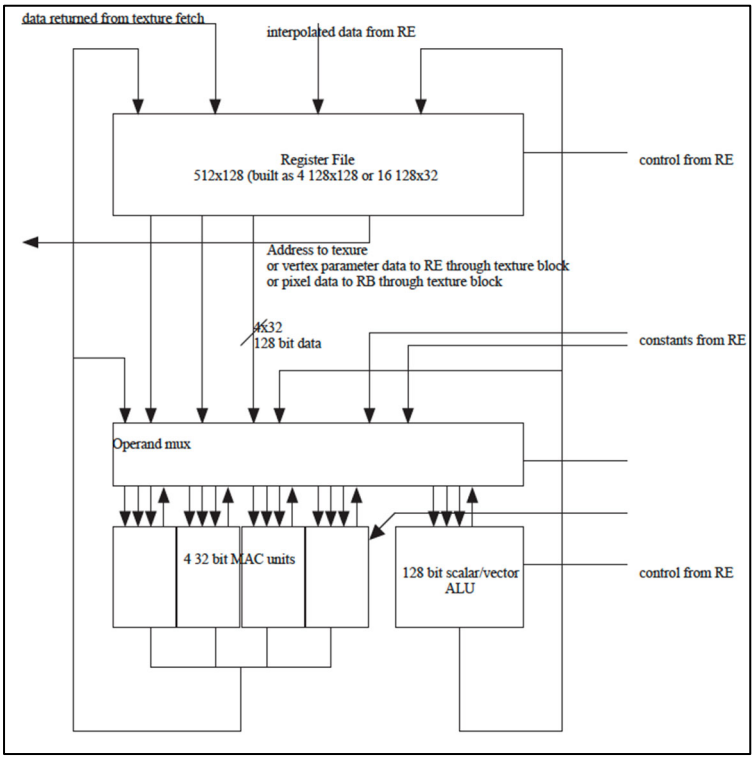
One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

99. Specifically, the R400 utilizes a unified shader, which comprises a SIMD/vector engine that performs pixel shader operations and vertex shader operations, to render 3D graphics. *See* 2013 (R400 Top Level Spec), at pp. 9-12. The unified processing pipeline of the R400, including its unified shader, performs 3D rendering by transforming vertices and determining the color of pixels. *See* EX. 2013 (R400 Top Level Spec), at pp. 10-12.

100. The passages above describe the unified shader as disclosed in the patent specification. The unified processing pipeline is a single pipeline that performs both pixel and vertex operations, achieve computation efficiency via load balancing among the different types of work, and space savings via area optimization. As described by Mr. Gruber and the above documentation, the unified shader is described as a vertex shader when it is performing vertex operations, and a pixel shader when performing pixel operations. *See* EX. 2113 (Gruber Dep.) at

100:19-101:1. Additionally, the following graphical depiction demonstrates the structure of the unified processing pipeline.



See EX. 2013 (R400 Top Level Spec), at p. 31.

101. The above disclosure also illustrates that the unified shader as conceived can be a single graphics processor component. *See Id*.

b. "a general purpose register block"

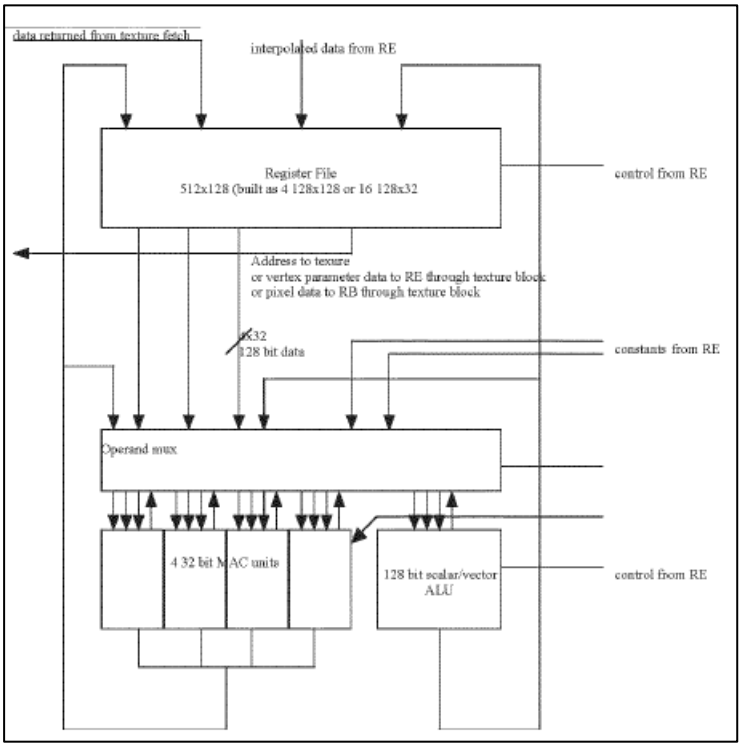
102. The first element of Claim 2 is "a general purpose register block." The specification of the '454 Patent discloses a shader that "includes a general purpose register block for storing at least the plurality of selected inputs . . . ." EX. 1001 at 3:4-5.

103. AMD documentation as of August 24, 2001, described this element.

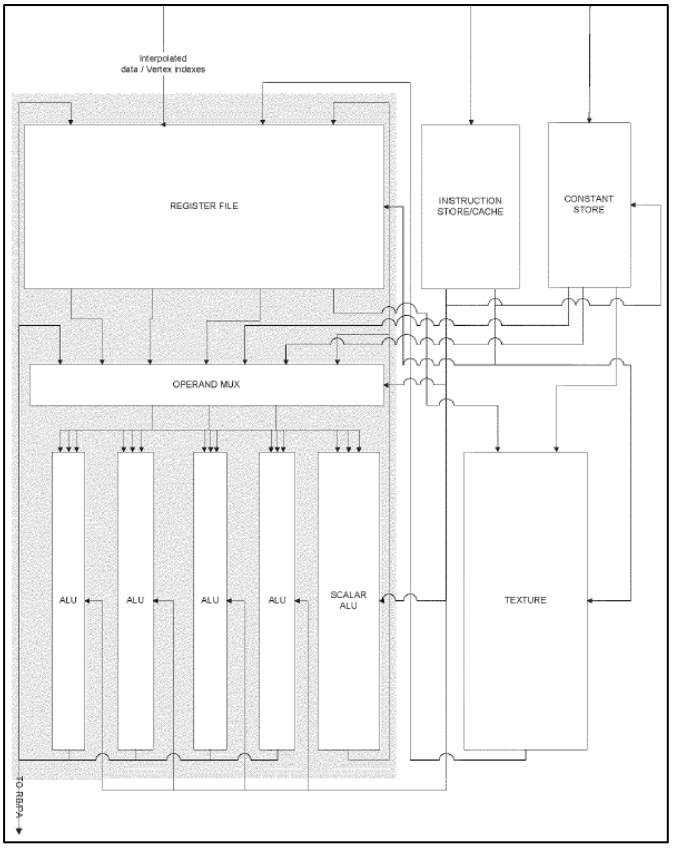
The R400 includes a general purpose register block. For example, the shader processor includes general purpose registers (GPRs) which are configured to store both vertex and pixel data, as initial input values, intermediate values, and final computation results.

> 1.1.1 *GPR*
>
> The general purpose registers are 128 bits wide, composed of four 32 bit values. Depending on the operation these values are interpreted at RGBA, or XYZW, or STQW, or UVQW, or YUVA, or.. to simplify matters the only two aliases used here are XYZW and RGBA.
>
> To hide the latency of memory acceses the shader pipe will switch between different vectors. This is the same as the idea of "microthreading" that some advanced CPU's are investigating. The large register file is split between the vectors executing in the shader pipe. The mangment of the shader register file is automatic, and not visible to a program executing on a vector, execept that a program is required to declare the number of GPRs it need to execute. The hardware will not start a vector until the required number of registers is available. There is a direct tradeoff between the number of registers each program/vector needs and the number of vectors than can be simultainiously resident. If there are too few vectors resident, then the latency of memory accesses can no longer be hidded and performance suffers.

See EX. 2012 (R400 Shader Processor), at p. 5.



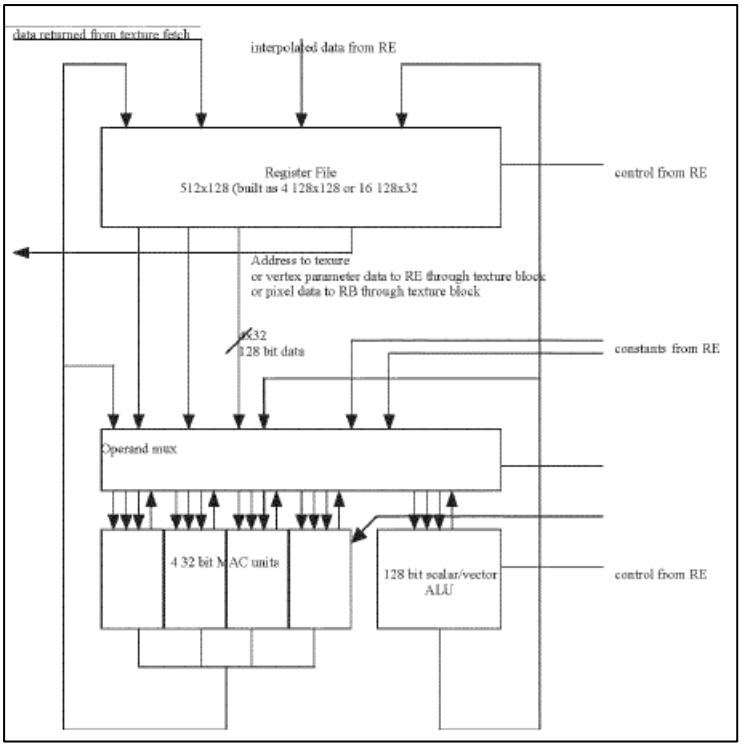
See EX. 2010 (R400 Sequencer Spec), at p. 7.



See EX. 2010 (R400 Sequencer Spec), at p. 11.

c. "a processor unit"

104. The second element of Claim 2 is a processor unit. AMD documentation as of August 24, 2001, described this element. The R400 includes a processor unit. For example, the R400 includes a unified programmable pipeline (e.g., unified shader) having multiple ALUs that make up a SIMD/vector engine performing pixel shader operations and vertex shader operations. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-12.



See EX. 2010 (R400 Sequencer Spec), at p. 7.

5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

The shader pipeline processes elements (pixels or vertices) in groups "vectors" of sixteen. R400 operates on tiles of 2x2 pixels or quad of pixels. There will be four sets of four shader pipes. For ease of reference and relative positioning of pixels within the quad that each set of shader blocks operates on, we name this sets as UL (upper left), Upper Right (upper right), LL (lower left) and LR (lower right). Please refer to the R400 Shader Processor Model (architectural specification) for an overall functionality of the shaders from the programmer's view-point.

See EX. 2012 (R400 Shader Processor), at p. 4.

d. "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"

105. The third element of Claim 2 recites "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."

106. Figure 5 of the '454 Patent shows a sequencer (99) along with its relationship to other components within the claimed graphics processor. EX. 1001 at Fig. 5.

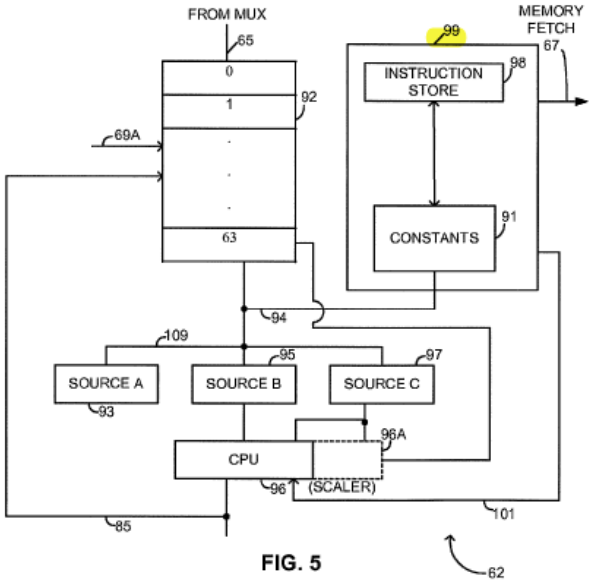


FIG. 5

107. The specification of the '454 Patent discloses the sequencer described

in the third element of Claim 2. In Figure 5, the patent discloses that "the unified shader 62 includes a general purpose register block 92, a plurality of source registers . . . and a sequencer 99." *Id.* at 4:30-34. Additional, "[i]f the next instruction to be executed is an arithmetic or logical instruction, the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block 92 into the appropriate source registers (93, 95, 97) for execution . . . ." *Id.* at 5:6-10.

108. The AMD documentation as of August 24, 2001, describes a sequencer having the same characteristics as the sequencer described in the '454 Patent. The R400 includes a sequencer.

> Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See EX. 2013 (R400 Top Level Spec), at p. 11.

> 8.15 RE – Raster Engine
>
> 8.15.1 *Description*
>
> The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

109. The sequencer, being part of the rasterizer, performs arbitration and is coupled to the general purpose register block and the processor unit.

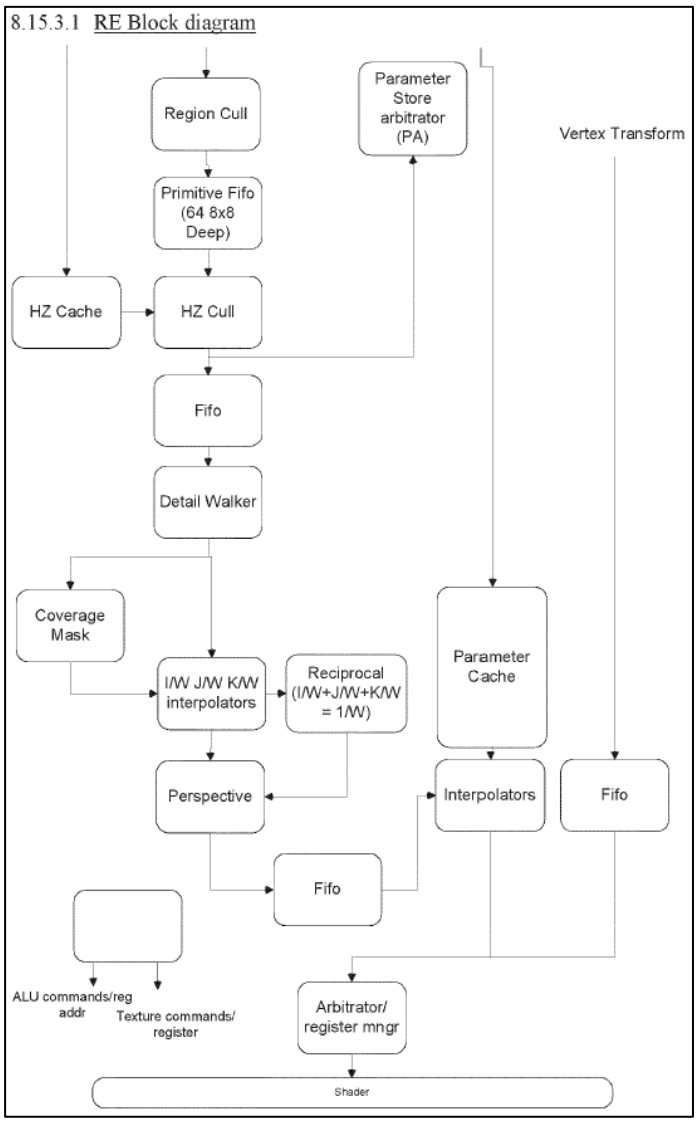
The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of ~~84~~ quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

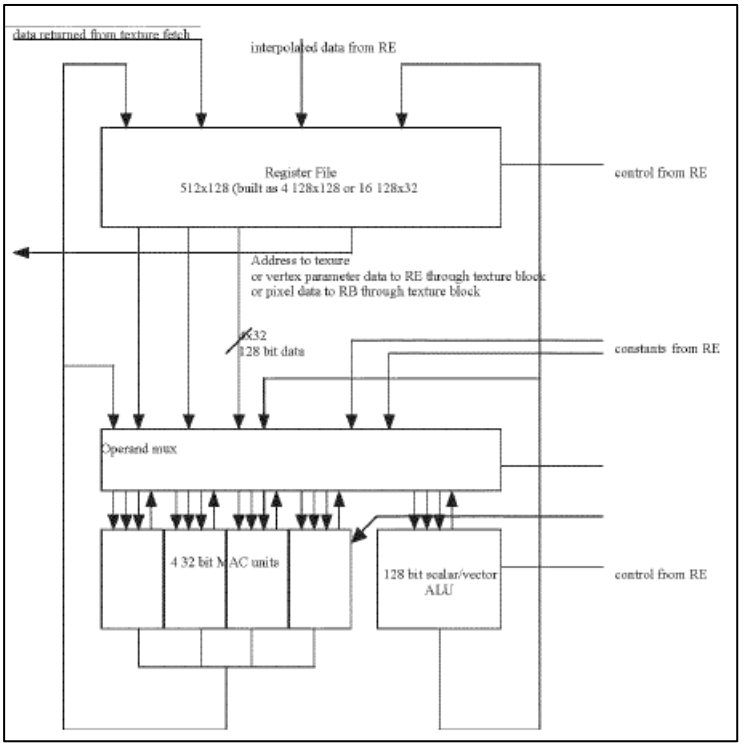
The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~aa~~ clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

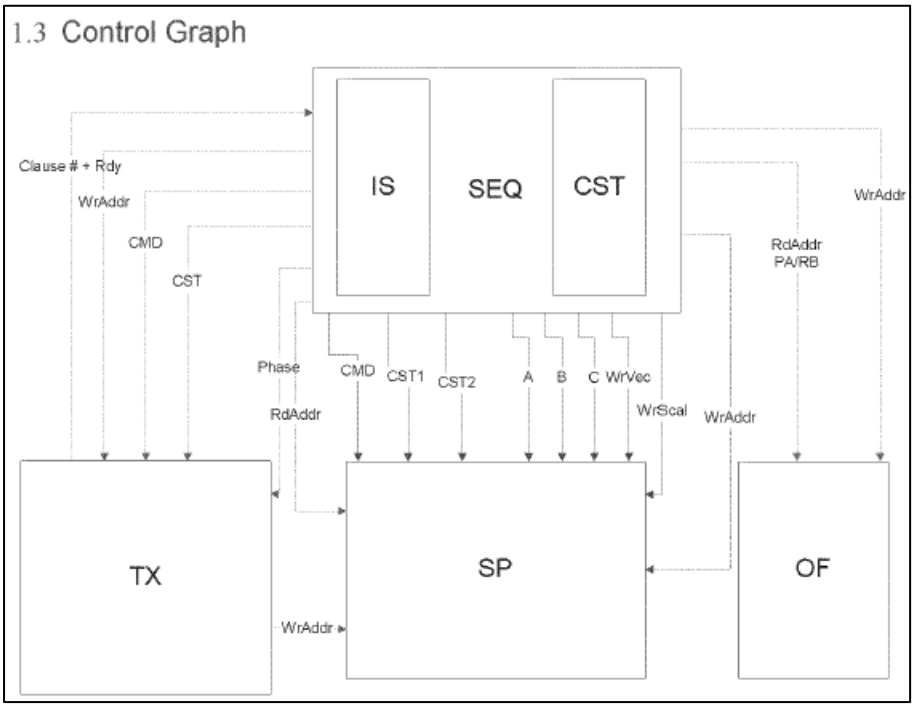
See EX. 2010 (R400 Sequencer Spec), at p. 4.



See EX. 2013 (R400 Top Level Spec), at p. 29.

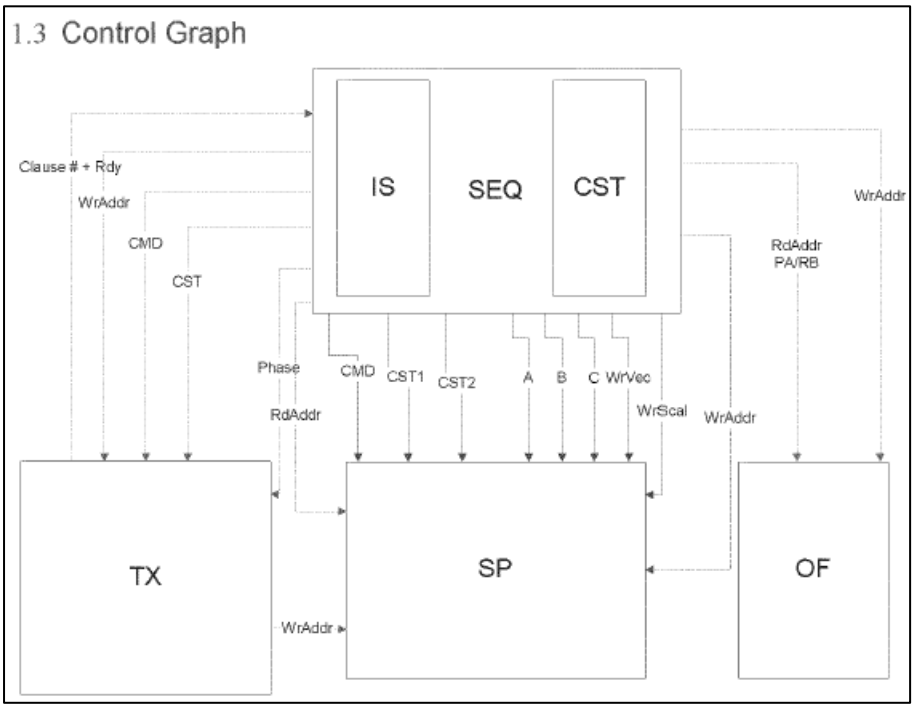


See EX 2010 (R400 Sequencer Spec), at p. 7.

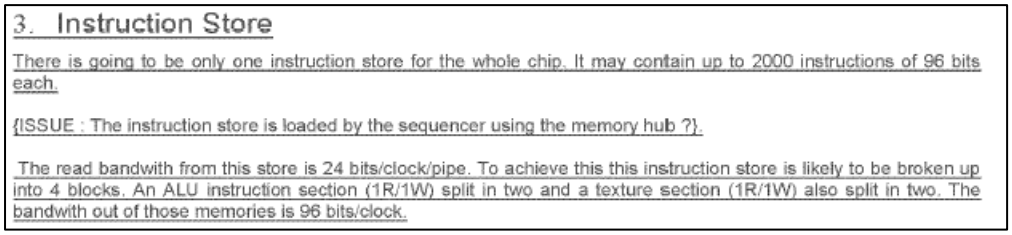


See Id. at p. 12.

110. The sequencer maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block. For example, the sequencer is coupled to an instruction store (IS), which stores vertex and pixel instructions.



See Id.



3. Instruction Store

There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

See Id.

1. PA sends a vector of 16 vertices (actually vertex indices – 32 bits/index for 512 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)

See Id. at p. 17.

13.1.2 *Sequencer Control of a Vector of Pixels*

1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.

See Id. at p. 18.

111. The operations are performed on data that is stored in the general purpose registers. For example, the unified shader uses a variable number of general purpose registers that are initialized with data. The ALU of the unified shader does simple math, conditional moves, and permutations on the registers that are initialized with data.

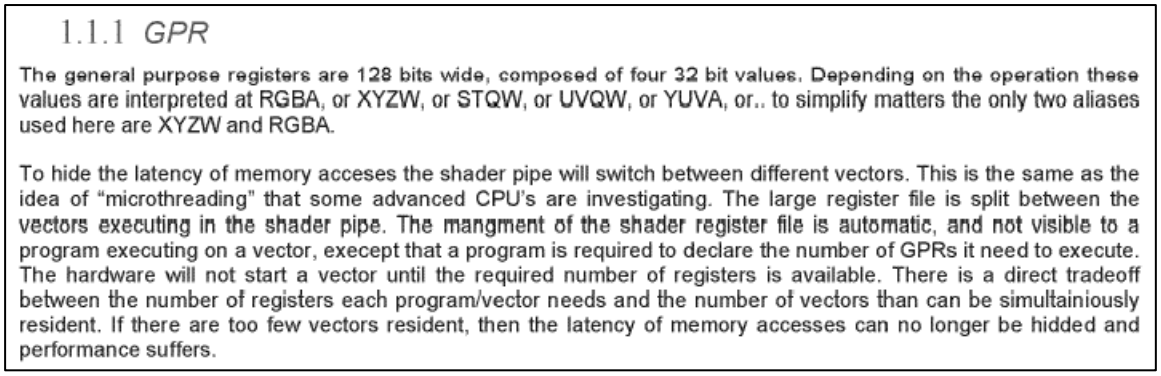
5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See 2013 (R400 Top Level Spec), at pp. 9-10.



1.1.1 *GPR*

The general purpose registers are 128 bits wide, composed of four 32 bit values. Depending on the operation these values are interpreted at RGBA, or XYZW, or STQW, or UVQW, or YUVA, or.. to simplify matters the only two aliases used here are XYZW and RGBA.

To hide the latency of memory acceses the shader pipe will switch between different vectors. This is the same as the idea of "microthreading" that some advanced CPU's are investigating. The large register file is split between the vectors executing in the shader pipe. The mangment of the shader register file is automatic, and not visible to a program executing on a vector, execept that a program is required to declare the number of GPRs it need to execute. The hardware will not start a vector until the required number of registers is available. There is a direct tradeoff between the number of registers each program/vector needs and the number of vectors than can be simultainiously resident. If there are too few vectors resident, then the latency of memory accesses can no longer be hidded and performance suffers.

See EX. 2012 (R400 Shader Processor), at p. 5.

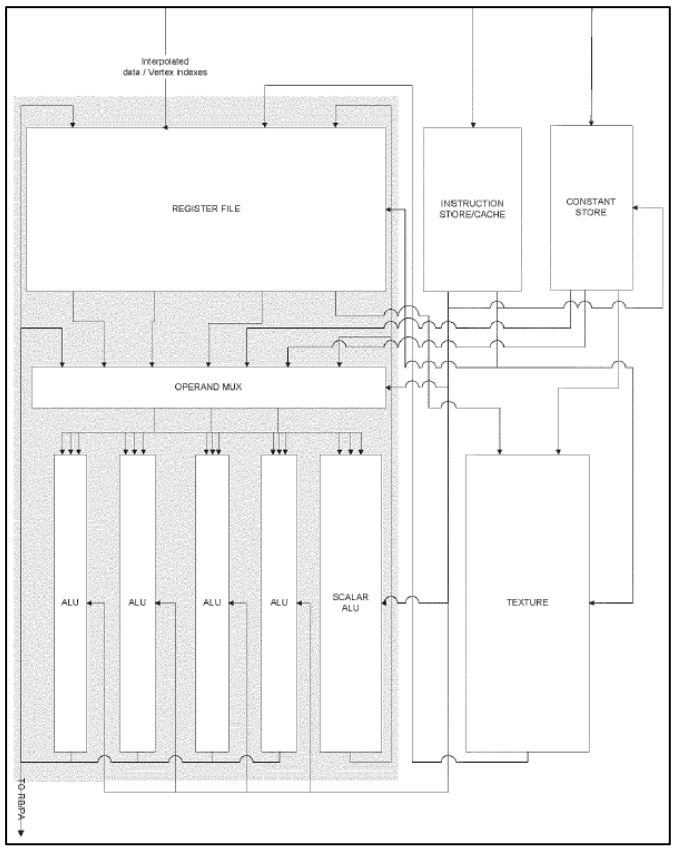
e. "wherein the processor unit executes instructions that generate a pixel color in response to selected data from the general purpose register block and generates vertex position and appearance data in response to selected data from the general purpose register block."

112. The final element of Claim 2 recites "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."

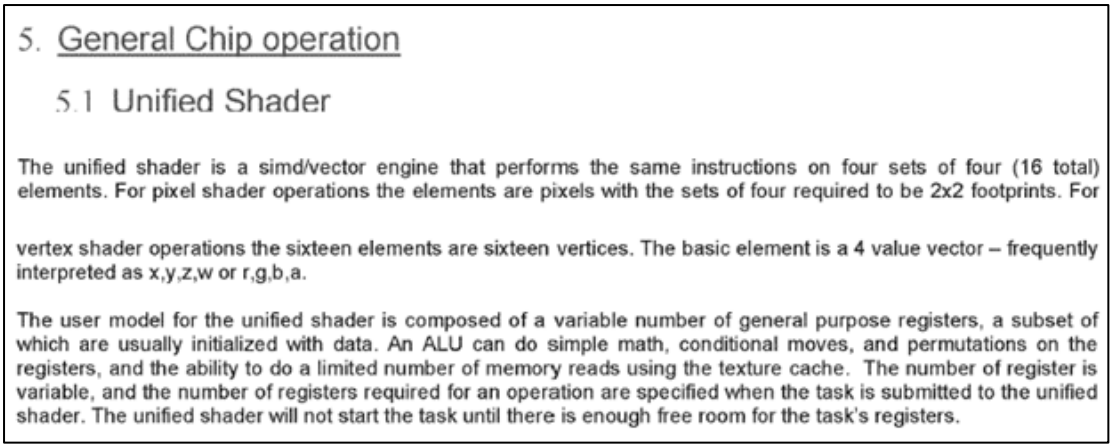
113. The specification of the '454 Patent discloses an exemplary embodiment of a processor "adapted to perform 32-bit floating point arithmetic operations as well as a complete series of logical operations on corresponding operands." EX. 1001 at 4:43-45. The patent further discloses that "[t]he general purpose register block 92 includes sixty four registers, or available entries, for storing the information . . . ." *Id.* at 4:34-36. Additionally, "the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block

92 into the appropriate source registers (93, 95, 97) for execution . . . .” *Id.* at 5:7-10.

114. The AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of the ’454 Patent. The processor unit of the R400 executes instructions. For example, the unified shader contains vector and scalar ALUs that execute instructions from the instruction store on data in the GPRs (e.g., register file).



See EX. 2010 (R400 Sequencer Spec), at p. 11.



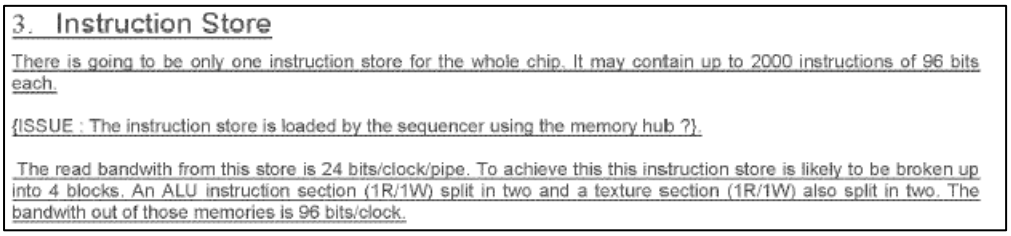
5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.



3. Instruction Store

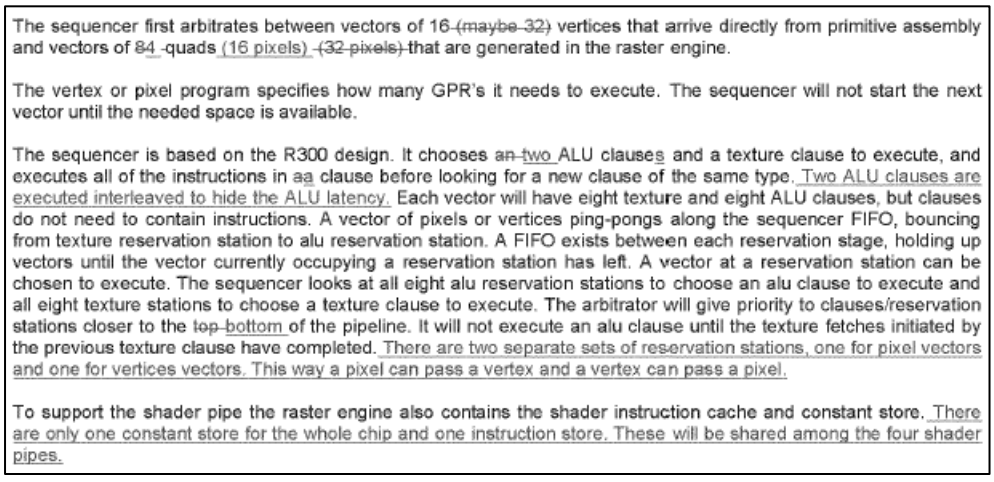
There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

See EX. 2010 (R400 Sequencer Spec), at p. 12.

115. The ALUs execute instructions on selected data. For example, the vertex or pixel program specifies how many GPRs it needs to execute.



The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of 84 quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

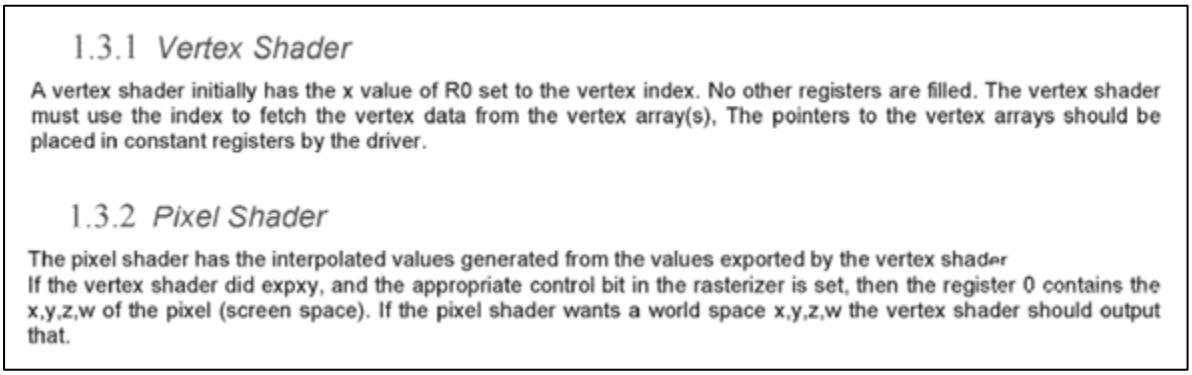
The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~a~~a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

See Id. at p. 4.

116. The vertex or pixel program is then executed by the ALUs of the shader processor, which operate on selected data that is initialized in the GPRs.



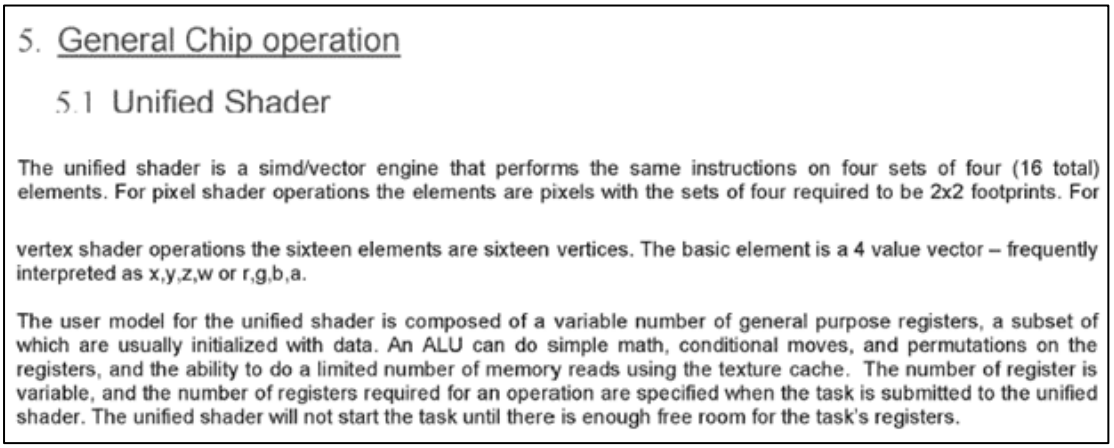
1.3.1 *Vertex Shader*

A vertex shader initially has the x value of R0 set to the vertex index. No other registers are filled. The vertex shader must use the index to fetch the vertex data from the vertex array(s), The pointers to the vertex arrays should be placed in constant registers by the driver.

1.3.2 *Pixel Shader*

The pixel shader has the interpolated values generated from the values exported by the vertex shader
If the vertex shader did expxy, and the appropriate control bit in the rasterizer is set, then the register 0 contains the x,y,z,w of the pixel (screen space). If the pixel shader wants a world space x,y,z,w the vertex shader should output that.

See EX. 2012 (R400 Shader Processor), at pp. 7-10.



5. General Chip operation

5.1 Unified Shader

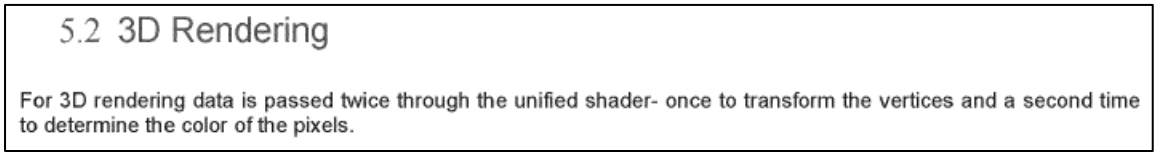
The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For

vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

117. The 3D rendering data that is selected from the GPRs and executed in the unified shader results in the generation of transformed vertices (e.g., position and appearance) and colored pixels.



5.2 3D Rendering

For 3D rendering data is passed twice through the unified shader- once to transform the vertices and a second time to determine the color of the pixels.

See Id. at pp. 9-10.

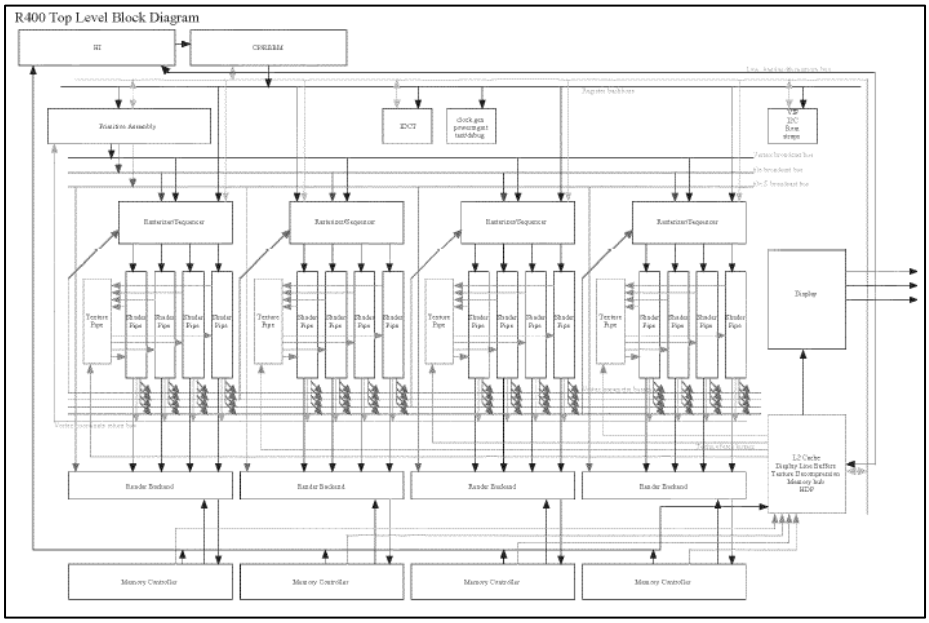
3. Claim 3

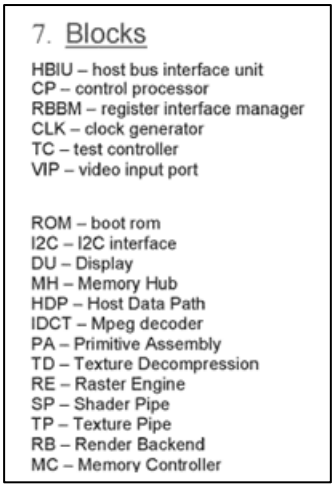
a. "A unified shader comprising:"

118. The preamble of Claim 3 recites "A unified shader comprising."

119. It is my opinion that the R400 is a unified shader.

120. AMD documentation as of August 24, 2001, described a unified shader having the same characteristics as the unified shader of the '454 Patent. For example, the Top Level Specification states that the R400 is a "standalone graphics chip product." *See* EX. 2013 (R400 Top Level Spec), at p. 6. Thus, the R400 is a graphics processor. Further, the Top Level Specification describes the R400 as having the following functional blocks including a shader processor and sequencer.





7. Blocks

HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller

See EX. 2013 (R400 Top Level Spec), at pp. 15-16.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See Id. at p. 11.

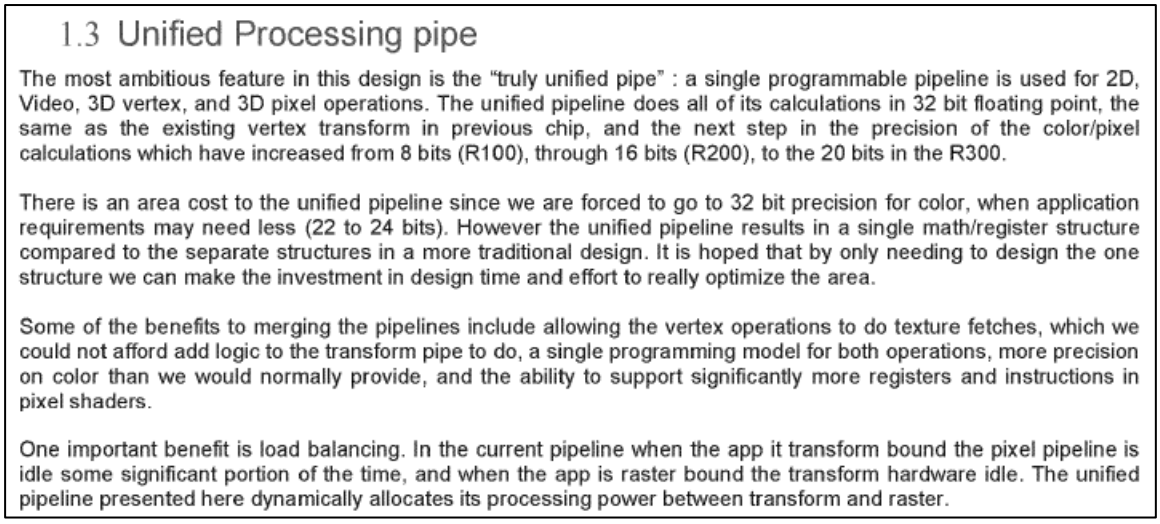
8.15 RE – Raster Engine

8.15.1 *Description*

The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

121. Moreover, the R400 implements a unified processing pipe to perform 3D vertex and pixel operations.



1.3 Unified Processing pipe

The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

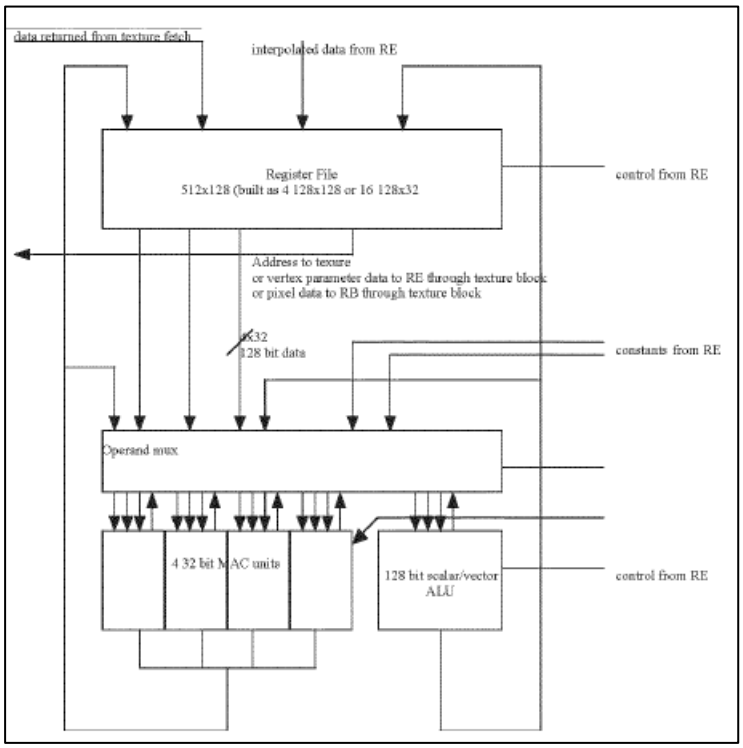
122. Specifically, the R400 utilizes a unified shader, which comprises a SIMD/vector engine that performs pixel shader operations and vertex shader operations, to render 3D graphics. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-10. The unified processing pipeline of the R400, including its unified shader, performs 3D rendering by transforming vertices and determining the color of pixels. *See Id.* at pp. 10-12.

b. "a processor unit operative to perform vertex calculation operations and pixel calculation operations"

123. The first element of Claim 3 recites "a processor unit operative to perform vertex calculation operations and pixel calculation operations."

124. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of the '454 Patent. The R400 includes a processor unit operative to perform vertex calculation operations and

pixel calculation operations. For example, the R400 includes a unified programmable pipeline (e.g., unified shader) having multiple ALUs that make up a SIMD/vector engine performing pixel shader operations and vertex shader operations. *See Id.* at pp. 9-12.



See EX. 2010, (R400 Sequencer Spec), at p. 7.

5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

> The shader pipeline processes elements (pixels or vertices) in groups "vectors" of sixteen. R400 operates on tiles of 2x2 pixels or quad of pixels. There will be four sets of four shader pipes. For ease of reference and relative positioning of pixels within the quad that each set of shader blocks operates on, we name this sets as UL (upper left), Upper Right (upper right), LL (lower left) and LR (lower right). Please refer to the R400 Shader Processor Model (architectural specification) for an overall functionality of the shaders from the programmer's view-point.

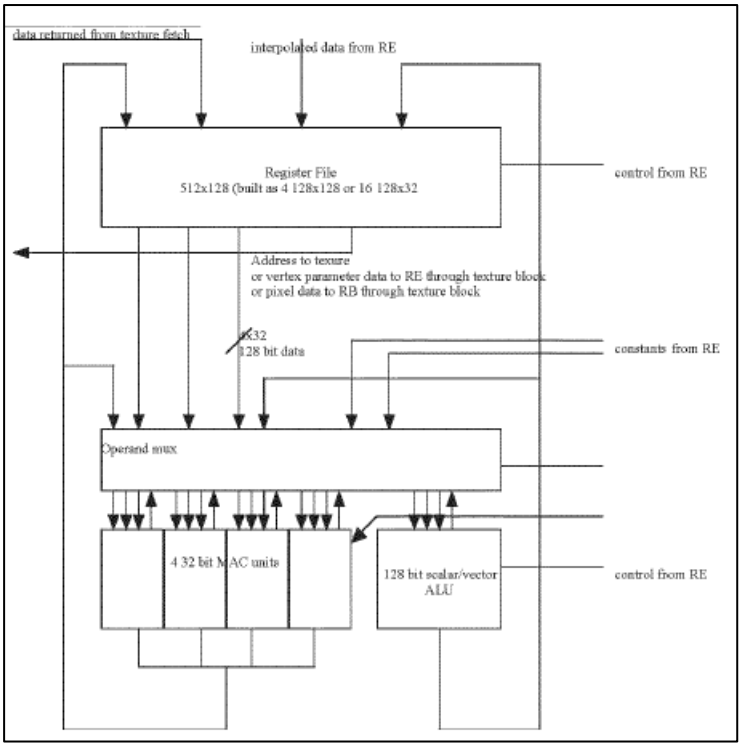
See EX. 2012 (R400 Shader Processor), at p. 4.

c. "shared resources, operatively coupled to the processor unit;"

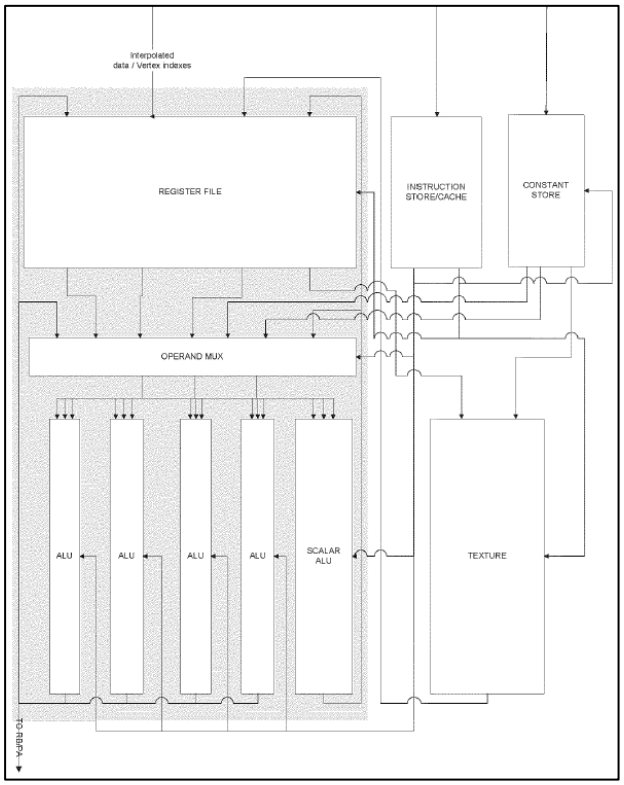
125. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit in this element of the '454 Patent. For example, the shader processor includes general purpose registers (GPRs) which are configured to store both vertex and pixel data, as initial input values, intermediate values, and final computation results.

> 1.1.1 *GPR*
>
> The general purpose registers are 128 bits wide, composed of four 32 bit values. Depending on the operation these values are interpreted at RGBA, or XYZW, or STQW, or UVQW, or YUVA, or.. to simplify matters the only two aliases used here are XYZW and RGBA.
>
> To hide the latency of memory acceses the shader pipe will switch between different vectors. This is the same as the idea of "microthreading" that some advanced CPU's are investigating. The large register file is split between the vectors executing in the shader pipe. The mangment of the shader register file is automatic, and not visible to a program executing on a vector, execept that a program is required to declare the number of GPRs it need to execute. The hardware will not start a vector until the required number of registers is available. There is a direct tradeoff between the number of registers each program/vector needs and the number of vectors than can be simultainiously resident. If there are too few vectors resident, then the latency of memory accesses can no longer be hidded and performance suffers.

See Id. at p. 5.



See EX. 2010, (R400 Sequencer Spec), at p. 7.

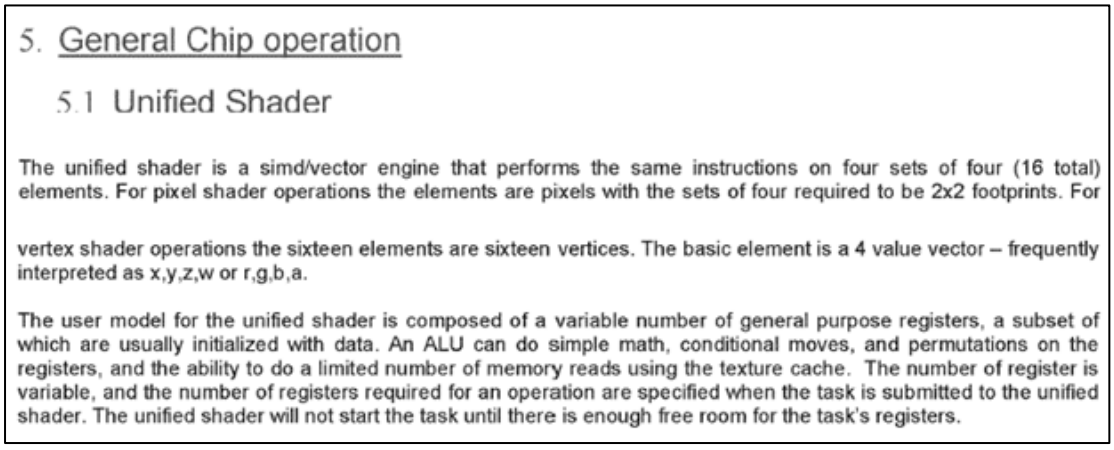


See Id. at p. 11.

d. "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."

126. The final element of Claim 3 recites "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."

127. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of the '454 Patent. For example, the ALUs of the unified shader initialize the GPRs with either vertex data or pixel data.



5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

128. The unified shader implements dynamic load balancing between

transform and raster.

> One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

129. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel calculation operations until enough shared resources become available, at which point the unified shader will perform vertex calculation operations.

> 2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
> - at this point the vector is removed from the Vertex FIFO
> - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 18.

> 4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 19.

> The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32-16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

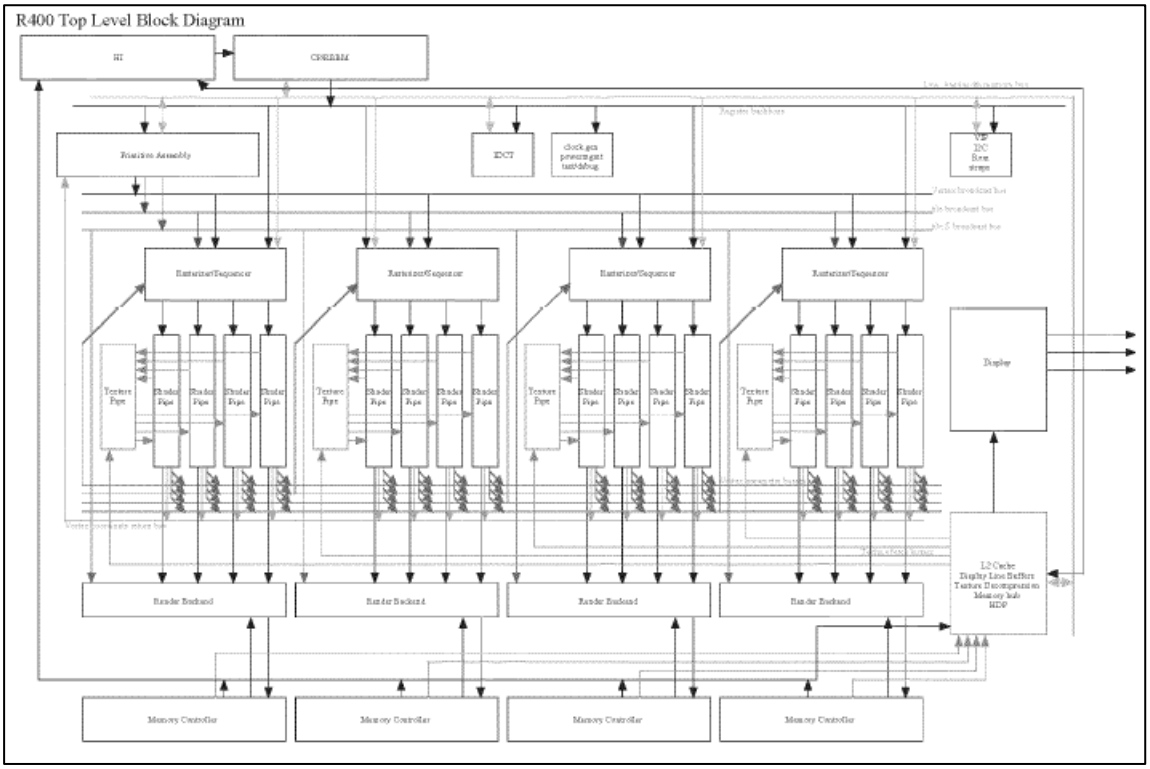
4. Claim 4

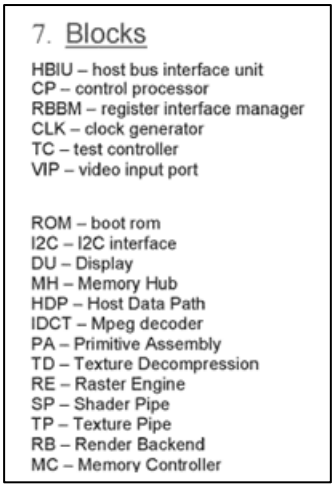
a. "A unified shader"

130. The preamble of Claim 4 recites "A unified shader, comprising."

131. It is my opinion that the R400 is a unified shader.

132. AMD documentation as of August 24, 2001, described a unified shader having the same characteristics as the unified shader of the '454 Patent. For example, the Top Level Specification states that the R400 is a "standalone graphics chip product." *See* EX. 2013 (R400 Top Level Spec), at p. 6. Thus, the R400 is a graphics processor. Further, the Top Level Specification describes the R400 as having the following functional blocks including a shader processor and sequencer.





7. Blocks

HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller

See EX. 2013 (R400 Top Level Spec), at pp. 15-16.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See EX. 2013 (R400 Top Level Spec), at p. 11.

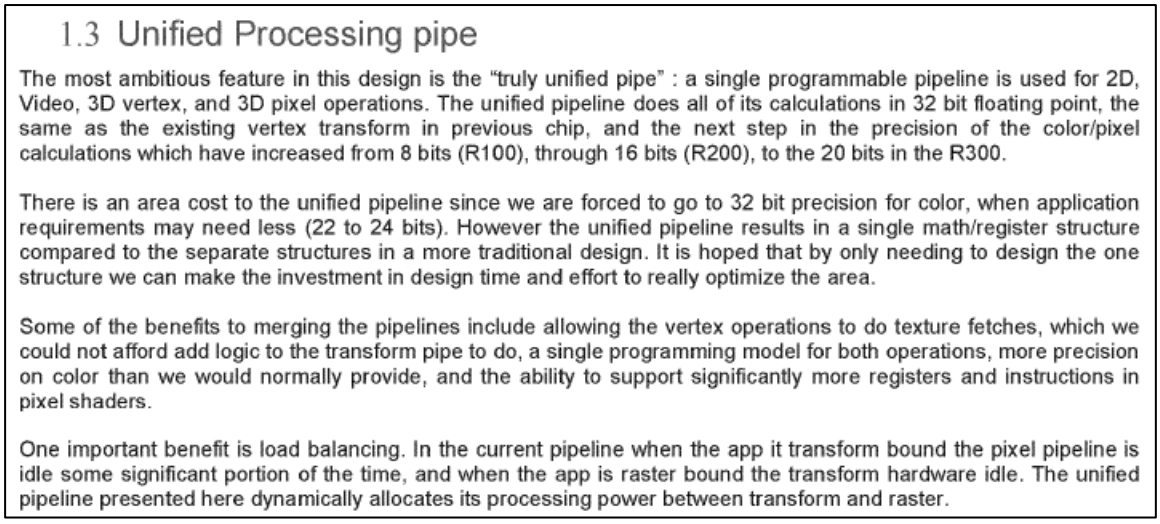
8.15 RE – Raster Engine

8.15.1 *Description*

The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

133. Moreover, the R400 implements a unified processing pipe to perform 3D vertex and pixel operations.



1.3 Unified Processing pipe

The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

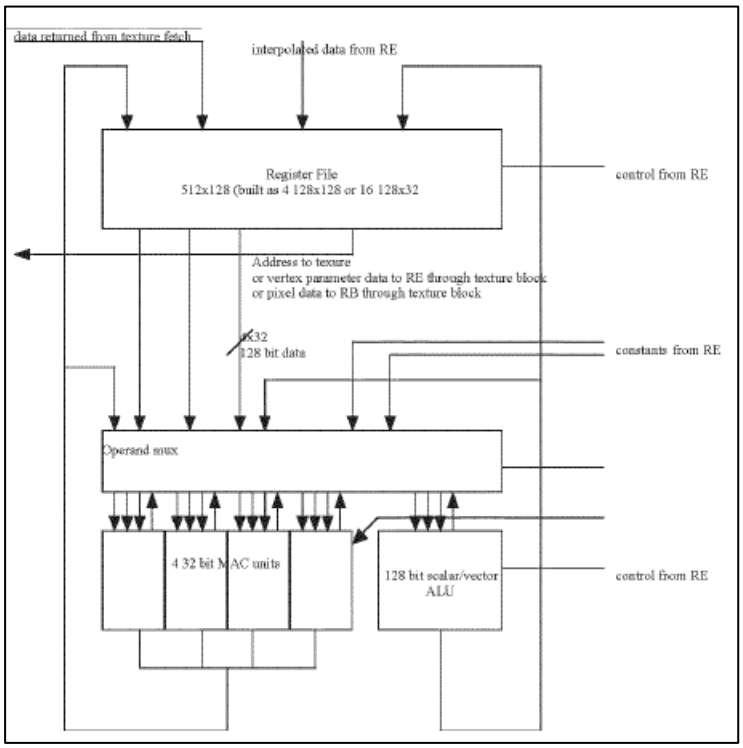
134. Specifically, the R400 utilizes a unified shader, which comprises a SIMD/vector engine that performs pixel shader operations and vertex shader operations, to render 3D graphics. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-10. The unified processing pipeline of the R400, including its unified shader, performs 3D rendering by transforming vertices and determining the color of pixels. *See Id*.

b. "a processor unit operative to perform vertex calculation operations and pixel calculation operations"

135. The second element of Claim 4 recites "a processor unit operative to perform vertex calculation operations and pixel calculation operations."

136. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of the '454 Patent. The R400 includes a processor unit operative to perform vertex calculation operations and

pixel calculation operations. For example, the R400 includes a unified programmable pipeline (e.g., unified shader) having multiple ALUs that make up a SIMD/vector engine performing pixel shader operations and vertex shader operations. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-12.



See EX. 2010 (R400 Sequencer Spec), at p. 7.

5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For

vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

> The shader pipeline processes elements (pixels or vertices) in groups "vectors" of sixteen. R400 operates on tiles of 2x2 pixels or quad of pixels. There will be four sets of four shader pipes. For ease of reference and relative positioning of pixels within the quad that each set of shader blocks operates on, we name this sets as UL (upper left), Upper Right (upper right), LL (lower left) and LR (lower right). Please refer to the R400 Shader Processor Model (architectural specification) for an overall functionality of the shaders from the programmer's view-point.

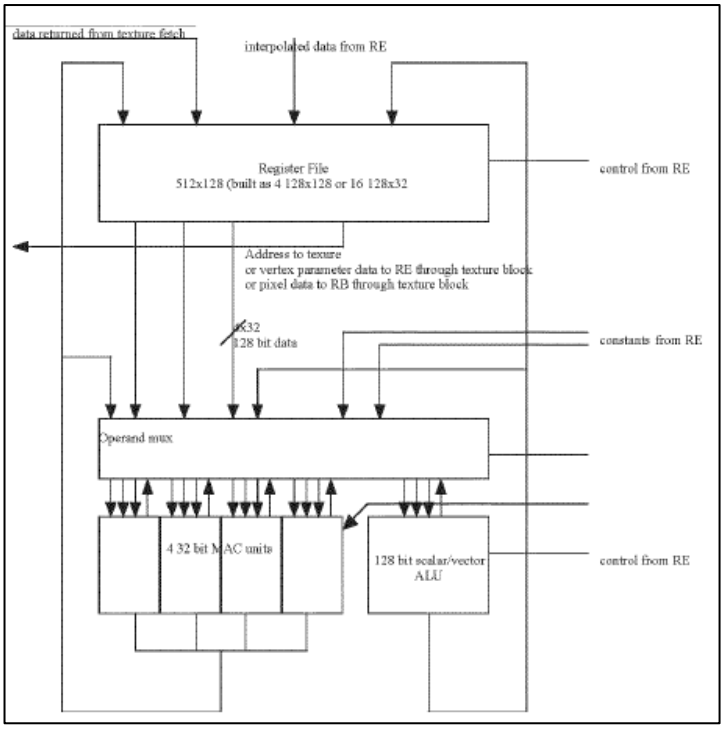
See EX. 2012 (R400 Shader Processor), at p. 4.

c. "shared resources, operatively coupled to the processor unit;"

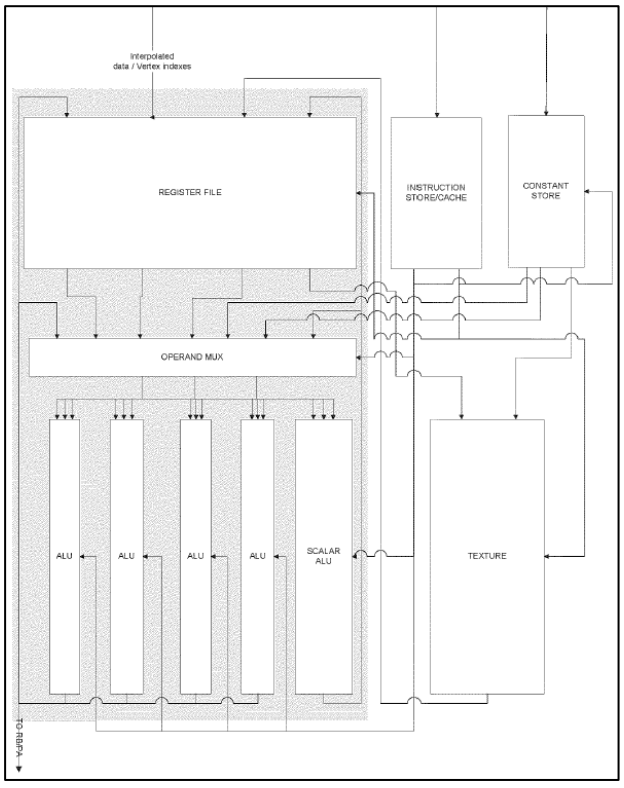
137. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of this element of the '454 Patent. For example, the shader processor includes general purpose registers (GPRs) which are configured to store both vertex and pixel data, as initial input values, intermediate values, and final computation results.

> 1.1.1 *GPR*
>
> The general purpose registers are 128 bits wide, composed of four 32 bit values. Depending on the operation these values are interpreted at RGBA, or XYZW, or STQW, or UVQW, or YUVA, or.. to simplify matters the only two aliases used here are XYZW and RGBA.
>
> To hide the latency of memory acceses the shader pipe will switch between different vectors. This is the same as the idea of "microthreading" that some advanced CPU's are investigating. The large register file is split between the vectors executing in the shader pipe. The mangment of the shader register file is automatic, and not visible to a program executing on a vector, execept that a program is required to declare the number of GPRs it need to execute. The hardware will not start a vector until the required number of registers is available. There is a direct tradeoff between the number of registers each program/vector needs and the number of vectors than can be simultainiously resident. If there are too few vectors resident, then the latency of memory accesses can no longer be hidded and performance suffers.

See EX. 2012 (R400 Shader Processor), at p. 5.



See EX. 2010 (R400 Sequencer Spec), at p. 7.

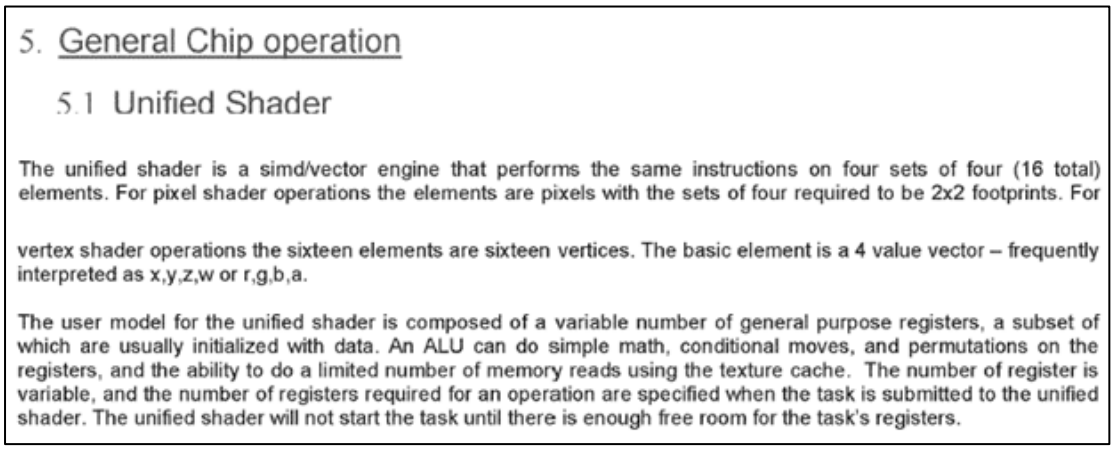


See EX. 2010 (R400 Sequencer Spec), at p. 11.

d. "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."

138. The final element of Claim 4 recites "a 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."

139. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of this element of the '454 Patent. For example, the ALUs of the unified shader initialize the GPRs with either vertex data or pixel data.



5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

140. The unified shader implements dynamic load balancing between transform and raster.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

141. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform vertex calculation operations as a priority until enough shared resources become available, at which point the unified shader will perform pixel calculation operations.

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 18.

4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 19.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32-16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

5. Claim 5

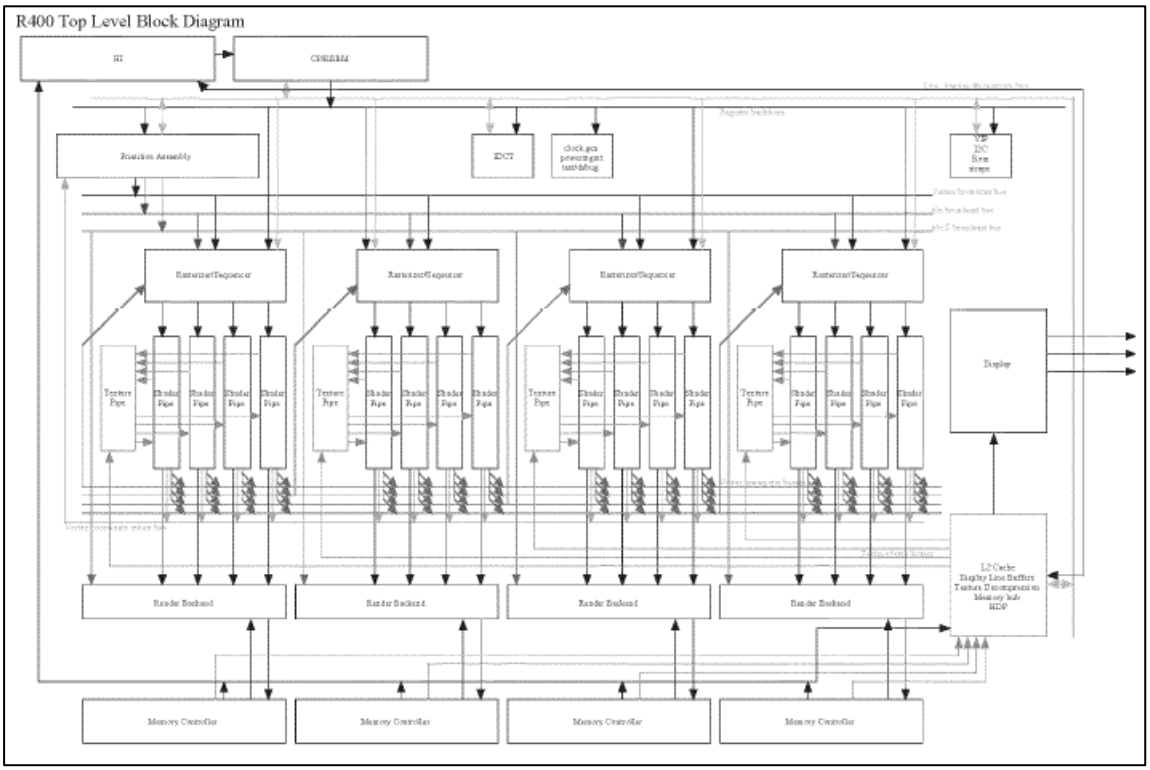
a. "A unified shader comprising:"

142. The preamble of Claim 5 recites "A unified, comprising."

143. It is my opinion that the R400 is a unified shader.

144. AMD documentation as of August 24, 2001, described a unified shader

having the same characteristics as the unified shader of the '454 Patent. For example, the Top Level Specification states that the R400 is a "standalone graphics chip product." *See* EX. 2013 (R400 Top Level Spec), at p. 6. Thus, the R400 is a graphics processor. Further, the Top Level Specification describes the R400 as having the following functional blocks including a shader processor and sequencer.



7. Blocks

HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller

See EX. 2013 (R400 Top Level Spec), at pp. 15-16.

> Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See Id. at p. 11.

> 8.15 RE – Raster Engine
>
> 8.15.1 *Description*
>
> The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

145. Moreover, the R400 implements a unified processing pipe to perform 3D vertex and pixel operations.

> 1.3 Unified Processing pipe
>
> The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.
>
> There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.
>
> Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.
>
> One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

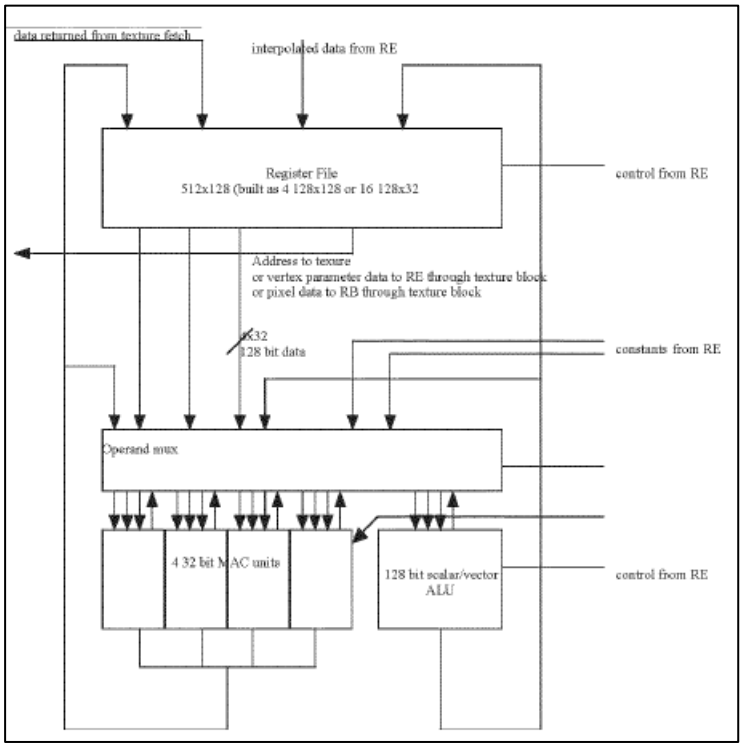
See EX. 2013 (R400 Top Level Spec), at p. 7.

146. Specifically, the R400 utilizes a unified shader, which comprises a SIMD/vector engine that performs pixel shader operations and vertex shader

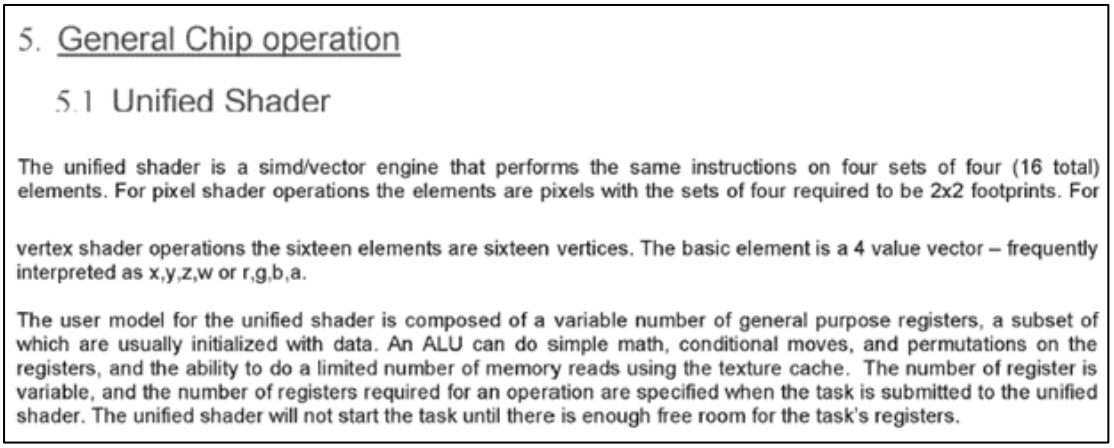
operations, to render 3D graphics. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-10. The unified processing pipeline of the R400, including its unified shader, performs 3D rendering by transforming vertices and determining the color of pixels. *See Id.* at pp. 10-12.

b. "a processor unit"

147. The first element of Claim 5 is "a processor unit." AMD documentation as of August 24, 2001, described a processor unit as characterized in the '454 Patent. Specifically, the R400 includes a processor unit. For example, the R400 includes a unified programmable pipeline (e.g., unified shader) having multiple ALUs that make up a SIMD/vector engine performing pixel shader operations and vertex shader operations. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-12.



See EX. 2010 (R400 Sequencer Spec), at p. 7.



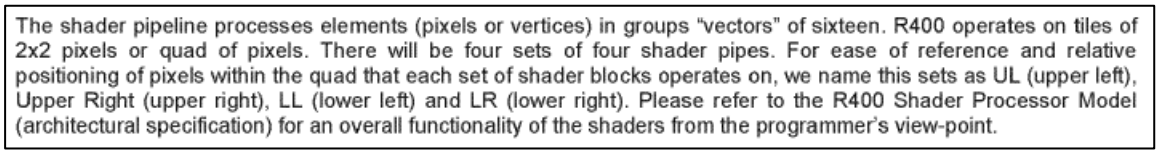
5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See Id. at pp. 9-10.



The shader pipeline processes elements (pixels or vertices) in groups "vectors" of sixteen. R400 operates on tiles of 2x2 pixels or quad of pixels. There will be four sets of four shader pipes. For ease of reference and relative positioning of pixels within the quad that each set of shader blocks operates on, we name this sets as UL (upper left), Upper Right (upper right), LL (lower left) and LR (lower right). Please refer to the R400 Shader Processor Model (architectural specification) for an overall functionality of the shaders from the programmer's view-point.

See Id. at p. 4.

c. "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."

148. The second element of Claim 5 recites "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."

149. The '454 Patent specifically discloses this element in its specification,

where it states "if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor 96 are continued, based on the instructions maintained in the instruction store 98. . . ." EX. 1001 at 5:41-48. Further, the patent discloses that the "sequencer 99 includes constants block 91 and an instruction store 98." *Id.* at 4:52-53.

150. AMD documentation as of August 24, 2001, described a sequencer having the same characteristics as the sequencer of the '454 Patent. The R400 includes a sequencer.

> Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See EX. 2013 (R400 Top Level Spec), at p. 11.

> 8.15 RE – Raster Engine
>
> 8.15.1 *Description*
>
> The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

151. The sequencer, being part of the rasterizer, performs arbitration and is coupled to the processor unit.

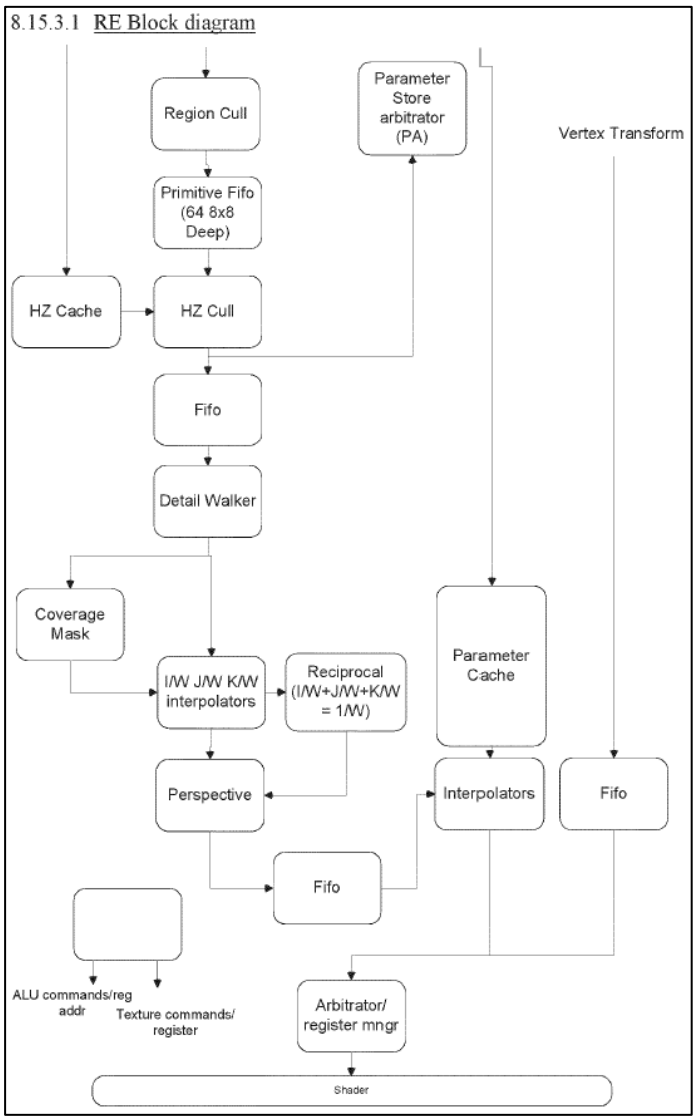
The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of ~~84~~ quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

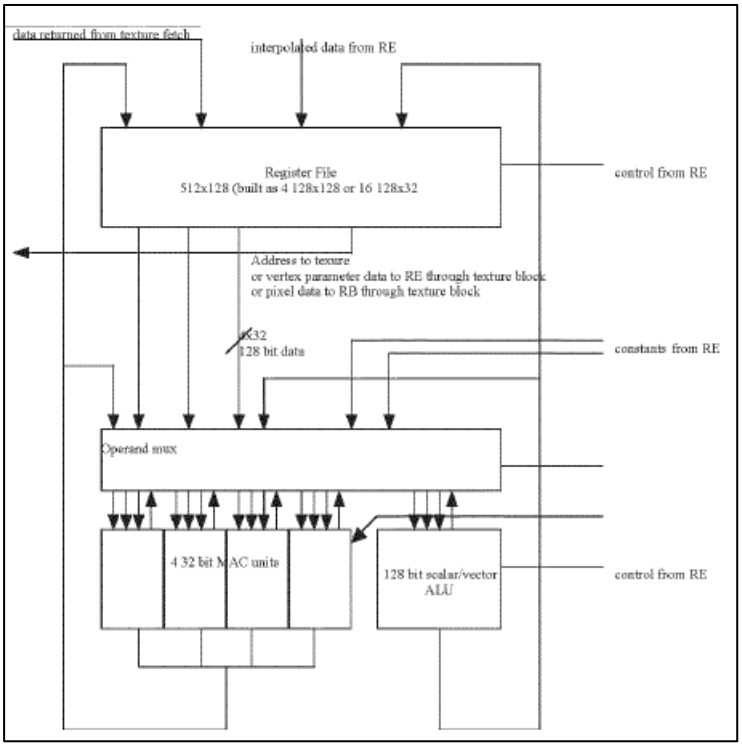
The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~aa~~ clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

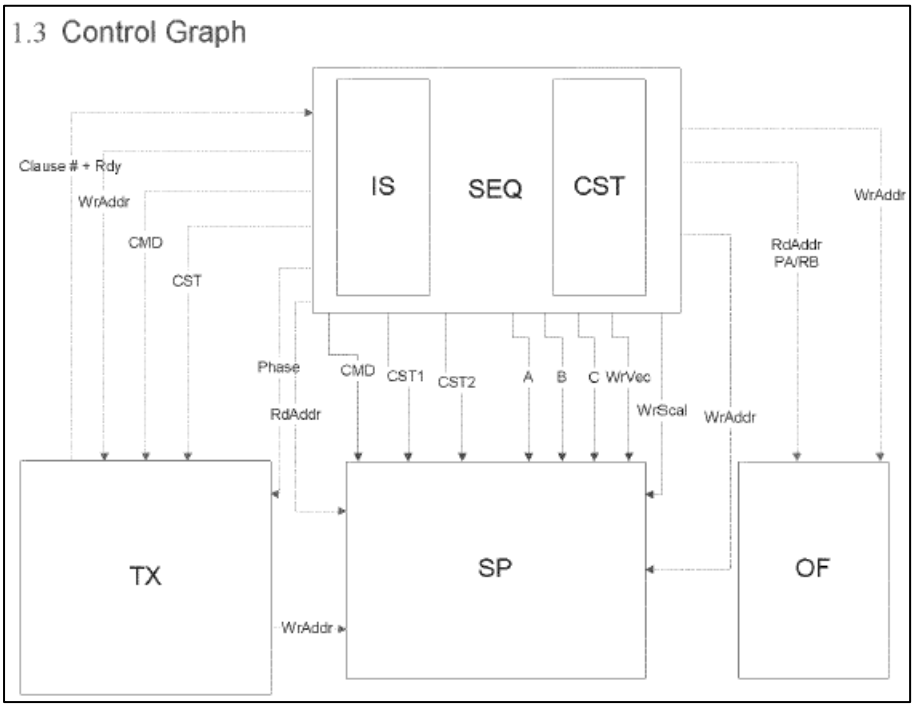
See EX. 2010 (R400 Sequencer Spec), at p. 4.



See EX. 2013 (R400 Top Level Spec), at p. 29.

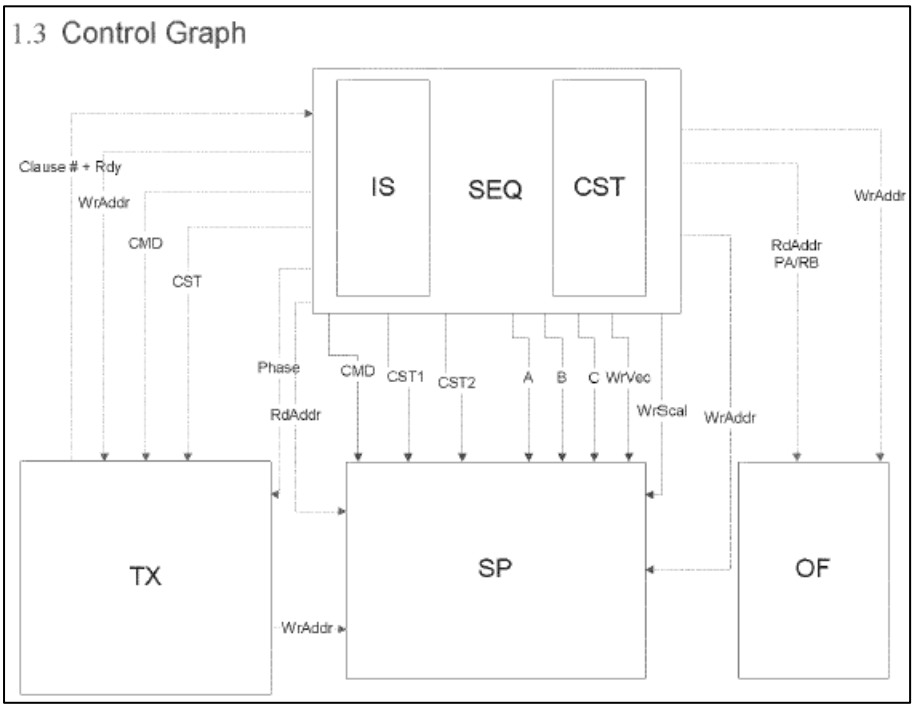


See EX. 2010 (R400 Sequencer Spec), at p. 7.



See EX. 2010 (R400 Sequencer Spec), at p. 12.

152. The sequencer maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block. For example, the sequencer is coupled to an instruction store (IS), which stores vertex and pixel instructions.



See Id.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

See Id.

1. PA sends a vector of 16 vertices (actually vertex indices – 32 bits/index for 512 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)

See Id. at p. 13.

13.1.2 *Sequencer Control of a Vector of Pixels*

1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.

See Id. at p. 18.

153. The operations are performed on data that is stored in the general purpose registers. For example, the unified shader uses a variable number of general purpose registers that are initialized with data. The ALU of the unified shader does simple math, conditional moves, and permutations on the registers that are initialized with data.

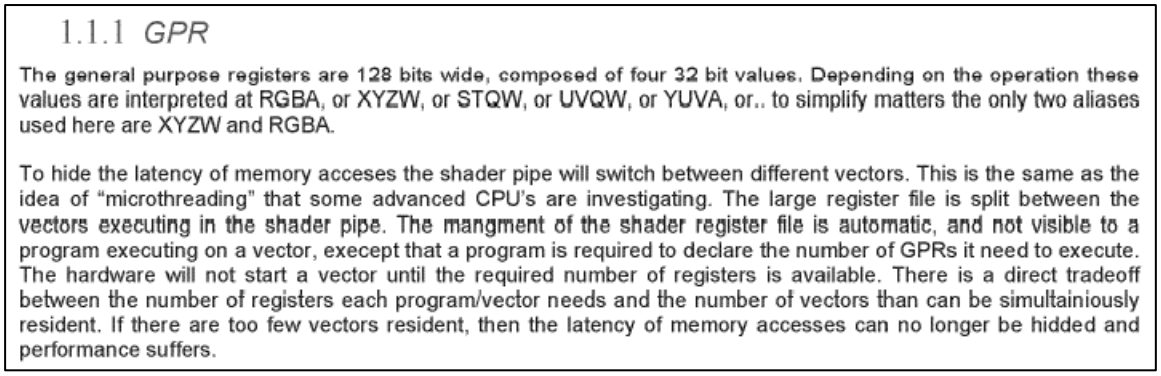
5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.



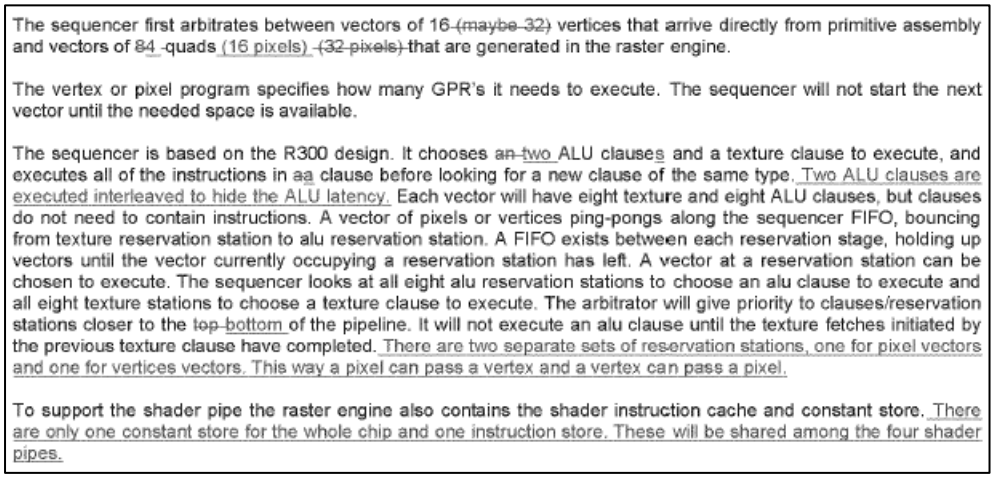
1.1.1 *GPR*

The general purpose registers are 128 bits wide, composed of four 32 bit values. Depending on the operation these values are interpreted at RGBA, or XYZW, or STQW, or UVQW, or YUVA, or.. to simplify matters the only two aliases used here are XYZW and RGBA.

To hide the latency of memory acceses the shader pipe will switch between different vectors. This is the same as the idea of "microthreading" that some advanced CPU's are investigating. The large register file is split between the vectors executing in the shader pipe. The mangment of the shader register file is automatic, and not visible to a program executing on a vector, execept that a program is required to declare the number of GPRs it need to execute. The hardware will not start a vector until the required number of registers is available. There is a direct tradeoff between the number of registers each program/vector needs and the number of vectors than can be simultainiously resident. If there are too few vectors resident, then the latency of memory accesses can no longer be hidded and performance suffers.

See EX. 2012 (R400 Shader Processor), at p. 5.

154. Operations will be executed on the data based on the availability of resources in the GPRs. Specifically, the vertex or pixel program specifies how many GPRs it needs to execute and the sequence will not start the next vector until the needed space is available.



The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of 84 quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in aa clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

See EX. 2010 (R400 Sequencer Spec), at p. 4.

6. Claim 6

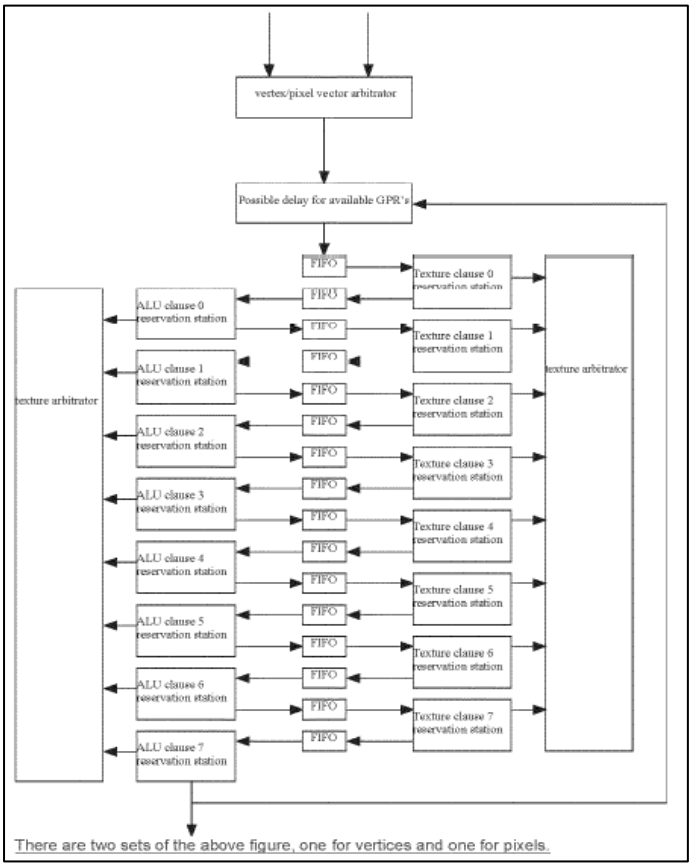
a. "The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory."

155. Claim 6 recites the shader of Claim 5, "wherein the sequencer further

includes circuitry operative to fetch data from a memory."

156. The specification of the '454 Patent discloses this element of Claim 6, stating "the sequencer 99 includes a constants block 91 and an instruction store 98. . . . The instruction store 98 further includes memory fetch instructions that, when executed, causes the unified shader 62 to fetch texture and other types of data, from memory 82 (FIG. 4A)." EX. 1001 at 4:52-63.

157. AMD documentation as of August 24, 2001, described a shader and sequencer having the same characteristics as the shader and sequencer of the '454 Patent. For example, the sequencer uses circuitry to select data from ALU or texture reservation stations for either vertex or pixel operations.



See EX. 2010 (R400 Sequencer Spec), at p. 5.

7. Claim 7

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

158. 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." The specification of the '454 Patent discloses this element of Claim 7, explaining that "the sequencer 99 determines whether the next instruction to be executed (from the instruction store 98) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction." EX. 1001 at 4:63-66. Figure 5 of the '454 Patent shows the selection circuit (e.g. sequencer 99) along with its relationship to the other components within the patent.

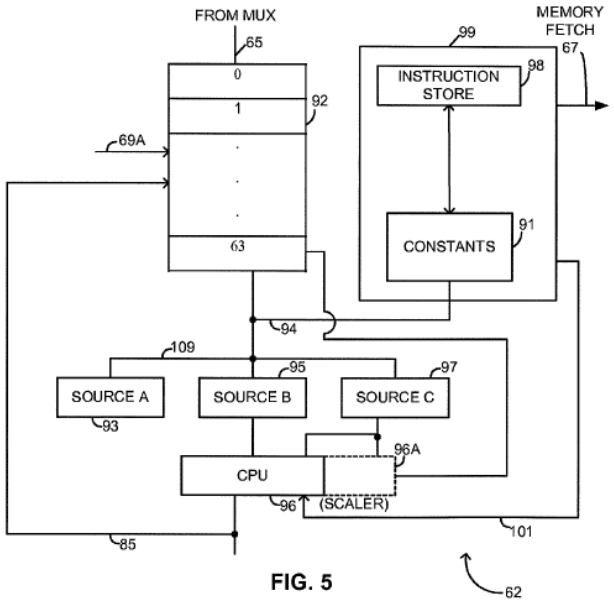
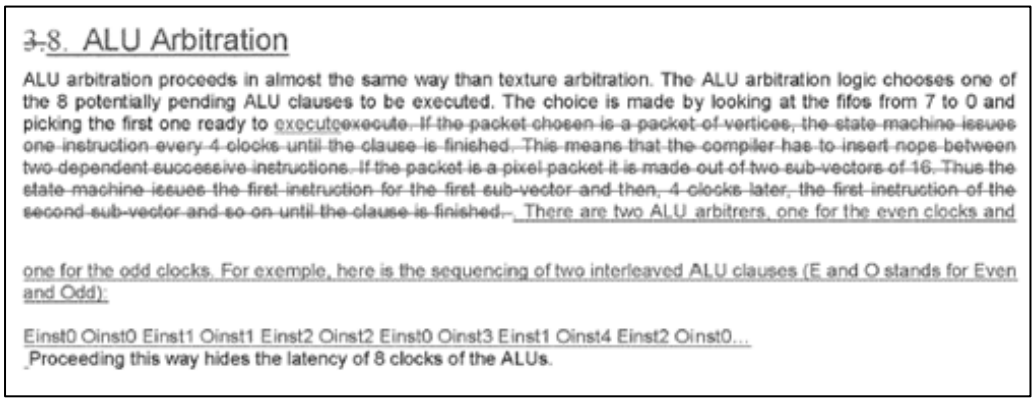


FIG. 5

159. The AMD documentation in the 2001 timeframe describes a shader having the same characteristics as the shader claimed in the '454 Patent. The R400 Sequencer, as explained above, is the arbiter, or selection circuit. The Sequencer performs ALU arbitration by choosing one of the 8 potentially pending ALU clauses.



3.8. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to executeexecute. If the packet chosen is a packet of vertices, the state machine issues one instruction every 4 clocks until the clause is finished. This means that the compiler has to insert nops between two dependent successive instructions. If the packet is a pixel packet it is made out of two sub-vectors of 16. Thus the state machine issues the first instruction for the first sub-vector and then, 4 clocks later, the first instruction of the second sub-vector and so on until the clause is finished.. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0...
Proceeding this way hides the latency of 8 clocks of the ALUs.

See EX. 2010 (R400 Sequencer Spec), at pp. 14-15.

160. The sequencer arbitrates between Pixel FIFO and Vertex FIFOs. When there are no vertices pending OR there is no space left in the register file for vertices the pixel FIFO is selected. *See Id.* at pp. 13-14.

161. The sequencer implements its arbitration scheme by generating control signals, for both pixel and vertex operations and instructions, in the form of control packets for either vector.

162. For vertex operations, the control packets contain the state pointer, the tag to the position cache and a register file base pointer. The control packets are sent to the reservation stations within the unified shader. From these control packets, the instructions for clauses are fetched from the global instruction store. The control

packet is then sent to a subsequent reservation station. All the instructions are received and the control packet is sent to the next reservation station. The control packet continues to travel down the path of reservation stations until all clauses have been executed.

13.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 16 vertices (actually vertex indices – 32 bits/index for 512 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
   - the 16 vertex indices are sent to the 16 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits (floating point format?) (what about compound indices) of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of texture clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

ATI Ex. No 2121
IPR2023-00922
Page 100 of 324

- the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
  - parameter data is sent to the Parameter Cache over a dedicated bus
  - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
  - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

See EX. 2010 (R400 Sequencer Spec), at pp. 17-18.

163. For pixel operations, the sequencer control starts with interpolation of parameters by sending barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator. The interpolator generates parameter values for each pixel. The sequencer control transfers the interpolated data to the SP register file over the RE_SP interface. The sequencer then constructs a control packet for the vector and sends it to the first reservation station. The control packet is then sent to the next reservation station. The control packet continues to travel down the reservation station path until all clauses have been executed.

13.1.2 *Sequencer Control of a Vector of Pixels*

1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector
   - after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer
   - the Parameter buffer is wide enough to source 3 vertices worth of a particular parameter in one cycle
   - at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA.
3. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source one quad's worth of barycentrics per cycle
4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
5. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
6. SEQ control starts with the interpolation of parameters (up to 16 per thread) by sending the barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator
   - P0i, P0j, and P0k (the value of P0 at each vertex) are loaded into the interpolator from the Parameter buffer
   - Q0 i, j, and k are loaded into the interpolator from the Pixel FIFO
   - The interpolator then generates the parameter value for each pixel in Q0 (Q0P0)
   - P0i, P0j, and P0k are sent to the interpolator for Q1 only if Q1 is from a different primitive; if Q1 is from the same primitive as Q0, then the P0i, P0j, and P0k values loaded for Q0 are held by the interpolator and reused for Q1
     - a "different_prim" control bit is passed with the barycentric data for each quad in the Pixel FIFO that indicates whether new parameter data needs to be loaded into the interpolator
   - Q1 i, j, and k are then loaded into the interpolator from the Pixel FIFO
   - The interpolator then generates the parameter value for each pixel in Q1 (Q1P0)
   - Q2P0 and Q3P0 are generated in a similar manner
   - The next set of parameter data - P1i, P1j, and P1k - is then loaded into the interpolator
   - Q0 i, j, and k now must be re-read from the Pixel FIFO – this means that the output of the Pixel FIFO loops through the top four entries on each read command until at the end a final "block_pop" signal is asserted, causing the top four sets of barycentric coordinates to finally be removed
   - so the order of parameter info generated is Q0P0, Q1P0, Q2P0, Q3P0, Q0P1, Q1P1, etc.
7. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
   - 16 pixels worth of interpolated parameter data is sent to the 16 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written with Q0P0 the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written with Q1P0 second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written with Q2P0 third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written with Q3P0 fourth cycle

8. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other informations (such as quad address for example) travels in a separate FIFO
9. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
10. all instructions of texture clause 0 are issued by TSM0
11. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for texture requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
12. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
13. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
14. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
      - it is sent to an output FIFO where it will be picked up by the render backend
      - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
15. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

ATI Ex. No 2121
IPR2023-00922
Page 102 of 324

See EX. 2010 (R400 Sequencer Spec), at pp. 18-19.

8. Claim 8

a. "The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected data."

164. 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."

165. Figure 5 of the '454 Patent (below) shows the processor (CPU) along with its relationship to other components within the claimed invention. EX. 1001 at Fig. 5.

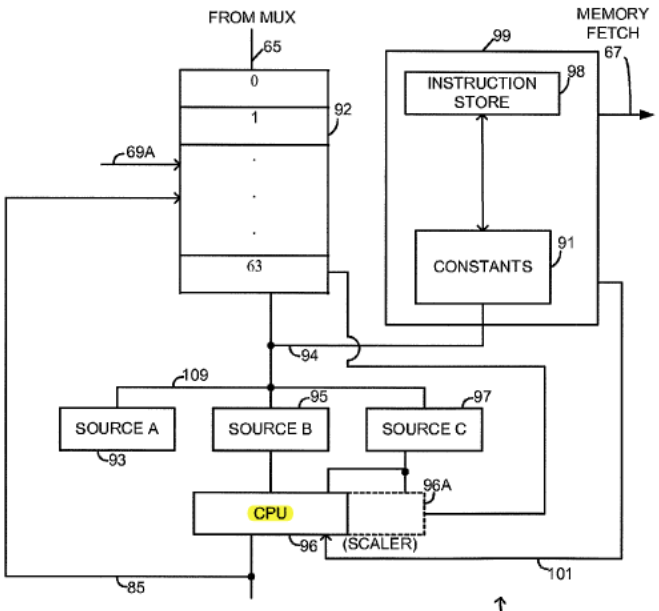
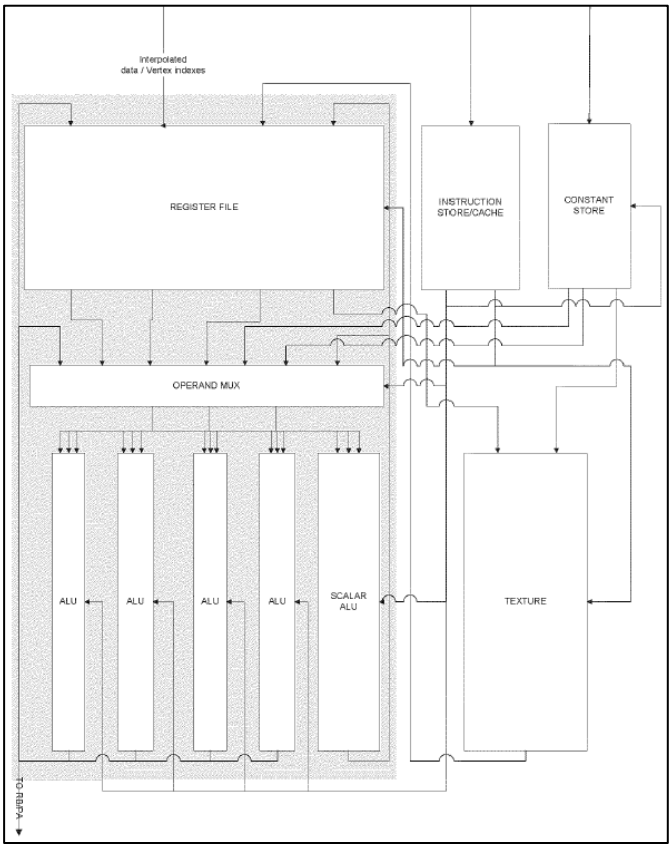


FIG. 5

166. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of the '454 Patent. The processor unit of the R400 executes instructions. For example, the unified shader

contains vector and scalar ALUs that execute instructions from the instruction store on data in the GPRs (e.g., register file).



See EX. 2010 (R400 Sequencer Spec), at p. 11.

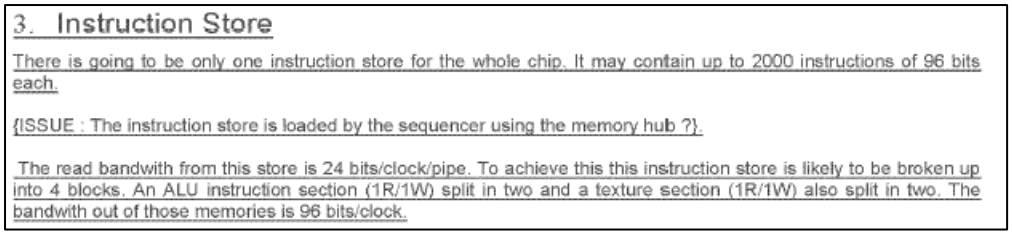
5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.



3. Instruction Store

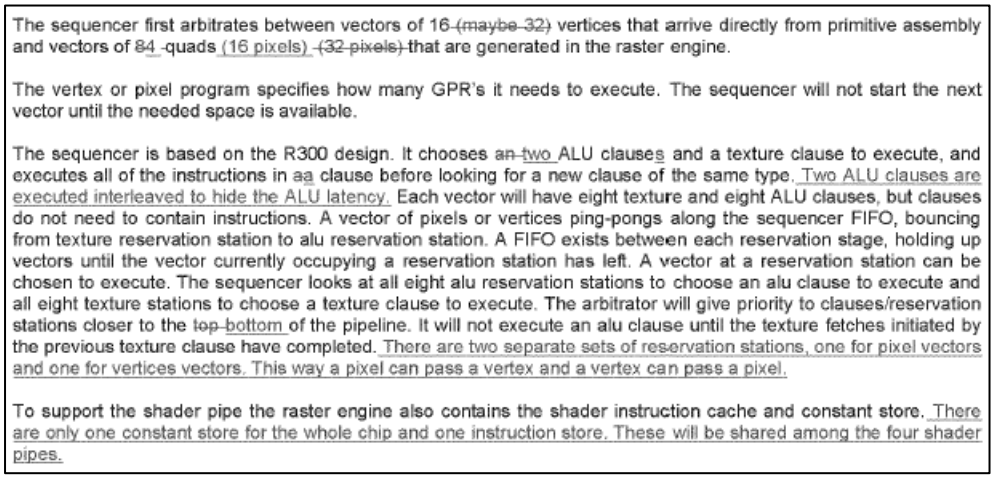
There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

See EX. 2010 (R400 Sequencer Spec), at p. 12.

167. The ALUs execute instructions on selected data. For example, the vertex or pixel program specifies how many GPRs it needs to execute.



The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of 84 ~~-~~quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

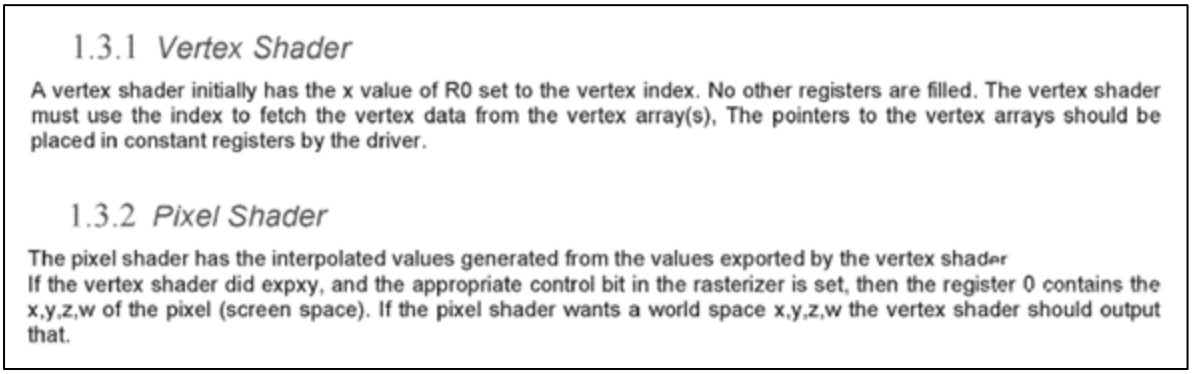
The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~a~~a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

See Id. at p. 4.

168. The vertex or pixel program is then executed by the ALUs of the shader processor, which operate on selected data that is initialized in the GPRs.



1.3.1 *Vertex Shader*

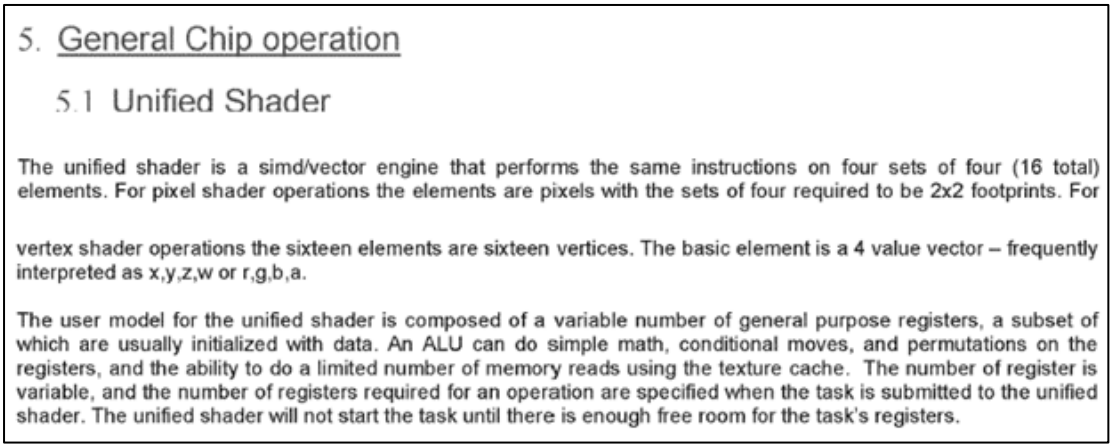
A vertex shader initially has the x value of R0 set to the vertex index. No other registers are filled. The vertex shader must use the index to fetch the vertex data from the vertex array(s), The pointers to the vertex arrays should be placed in constant registers by the driver.

1.3.2 *Pixel Shader*

The pixel shader has the interpolated values generated from the values exported by the vertex shader
If the vertex shader did expxy, and the appropriate control bit in the rasterizer is set, then the register 0 contains the x,y,z,w of the pixel (screen space). If the pixel shader wants a world space x,y,z,w the vertex shader should output that.

ATI Ex. No 2121
IPR2023-00922
Page 105 of 324

See EX. 2012 (R400 Shader Processor), at pp. 7-10.



5. General Chip operation

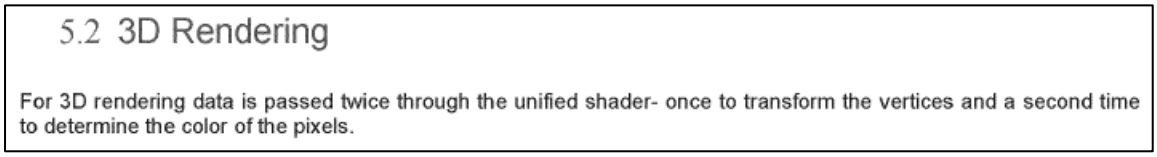
5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

169. The 3D rendering data that is selected from the GPRs and executed in the unified shader results in the generation of transformed vertices (e.g., position and appearance) and colored pixels.



5.2 3D Rendering

For 3D rendering data is passed twice through the unified shader- once to transform the vertices and a second time to determine the color of the pixels.

See Id.

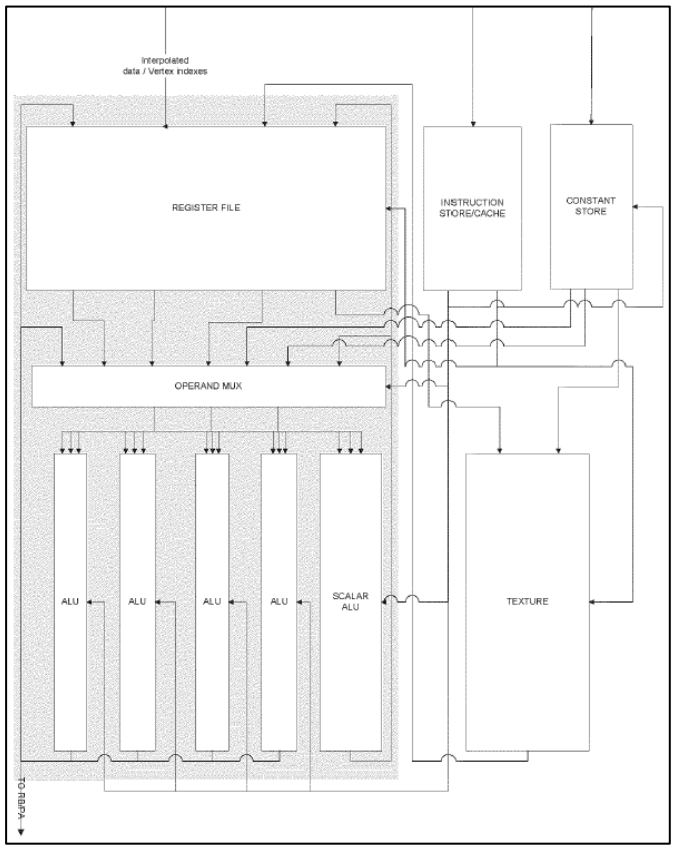
9. Claim 9

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

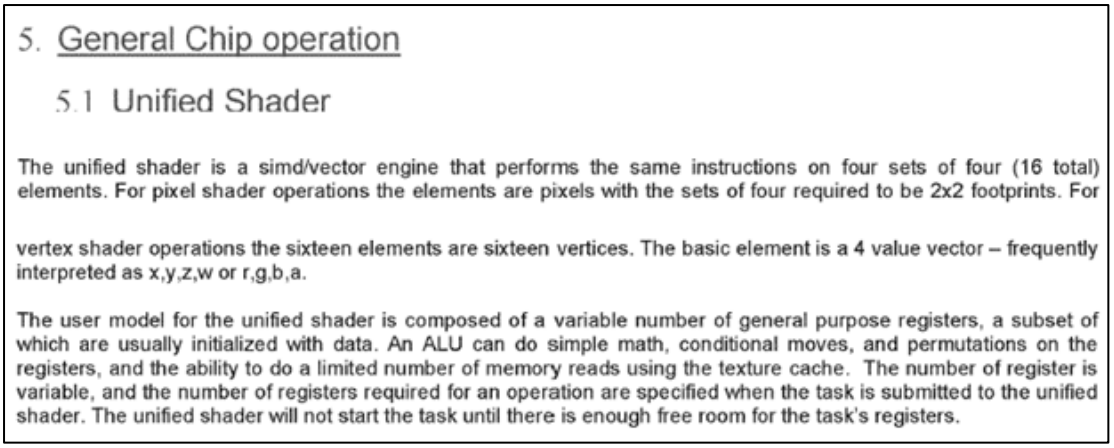
170. Claim 9 recites the shader of Claim 5, "wherein the processor unit generates vertex position and appearance data in response to a selected data."

171. AMD documentation as of August 24, 2001, described a processor unit

having the same characteristics as the processor unit of the '454 Patent. The processor unit of the R400 executes instructions. For example, the unified shader contains vector and scalar ALUs that execute instructions from the instruction store on data in the GPRs (e.g., register file).



See EX. 2010 (R400 Sequencer Spec), at p. 11.



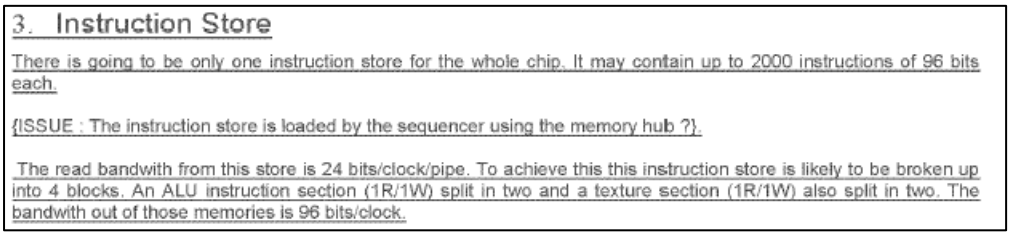
5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.



3. Instruction Store

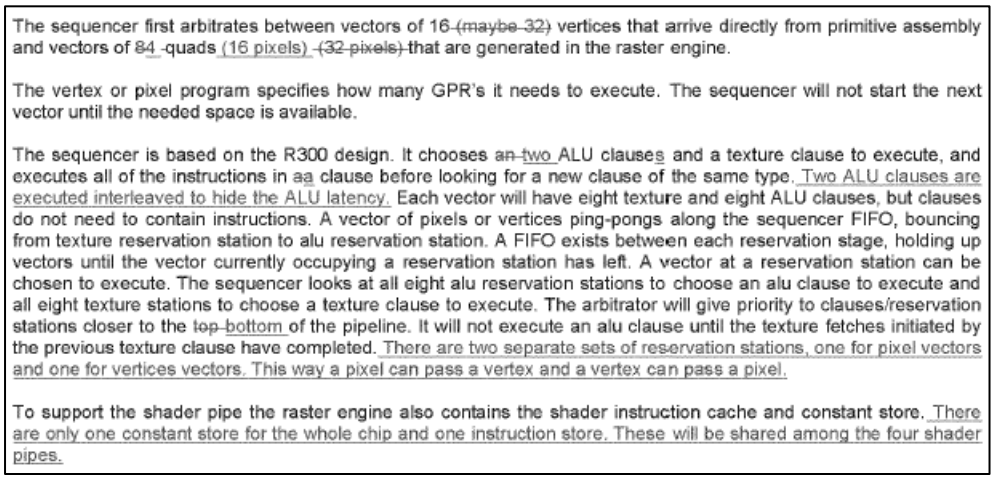
There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

See EX. 2010 (R400 Sequencer Spec), at p. 12.

172. The ALUs execute instructions on selected data. For example, the vertex or pixel program specifies how many GPRs it needs to execute.



The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of 84 quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

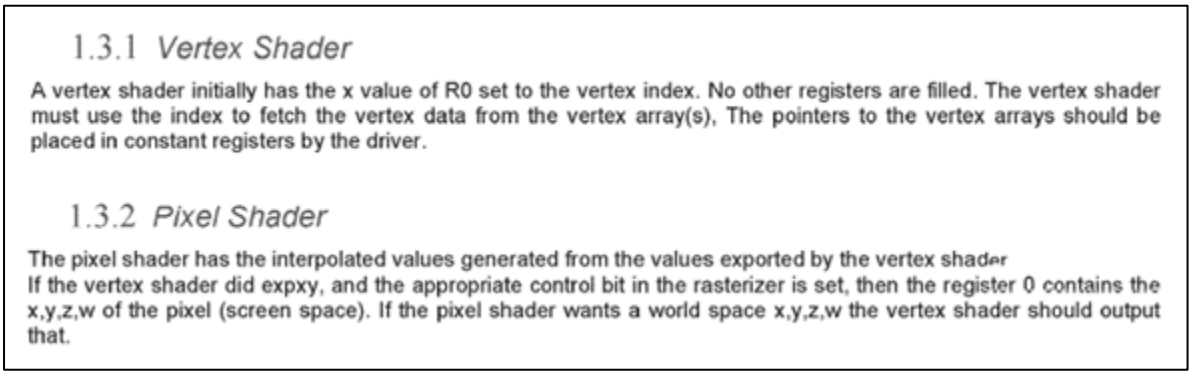
The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an two~~ ALU clauses and a texture clause to execute, and executes all of the instructions in aa clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

See EX. 2010 (R400 Sequencer Spec), at p. 4.

173. The vertex or pixel program is then executed by the ALUs of the shader processor, which operate on selected data that is initialized in the GPRs.



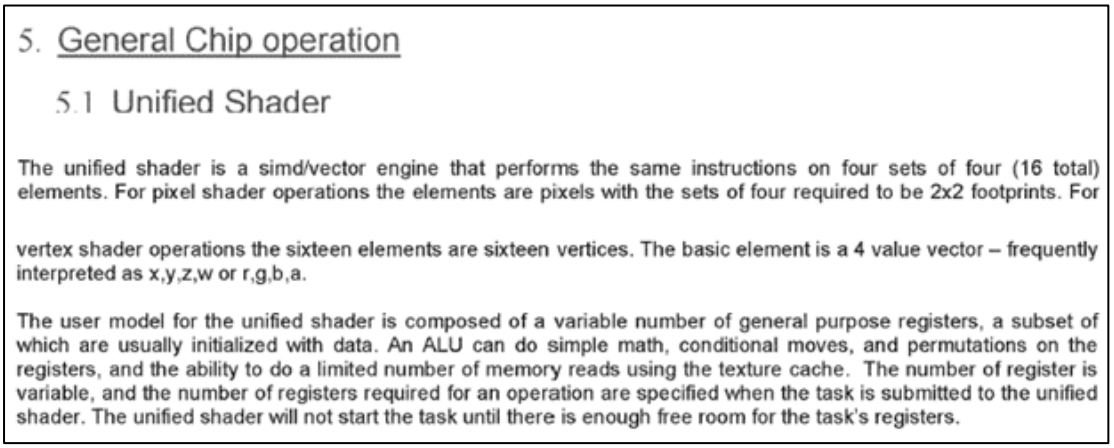
1.3.1 *Vertex Shader*

A vertex shader initially has the x value of R0 set to the vertex index. No other registers are filled. The vertex shader must use the index to fetch the vertex data from the vertex array(s), The pointers to the vertex arrays should be placed in constant registers by the driver.

1.3.2 *Pixel Shader*

The pixel shader has the interpolated values generated from the values exported by the vertex shader
If the vertex shader did expxy, and the appropriate control bit in the rasterizer is set, then the register 0 contains the x,y,z,w of the pixel (screen space). If the pixel shader wants a world space x,y,z,w the vertex shader should output that.

See EX. 2012 (R400 Shader Processor), at pp. 7-10.



5. General Chip operation

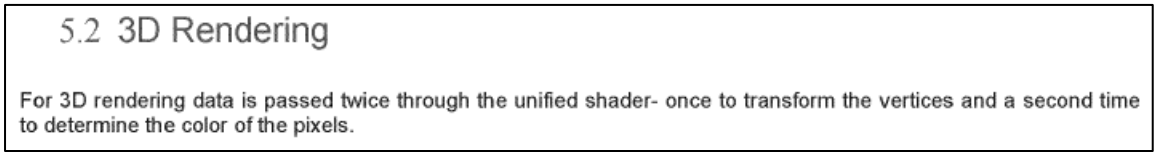
5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

See EX. 2013 (R400 Top Level Spec), at pp. 9-10.

174. The 3D rendering data that is selected from the GPRs and executed in the unified shader results in the generation of transformed vertices (e.g., position and appearance) and colored pixels.



5.2 3D Rendering

For 3D rendering data is passed twice through the unified shader- once to transform the vertices and a second time to determine the color of the pixels.

See Id.

10. Claim 10

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

175. Claim 10 recites the shader of Claim 7, "wherein the control signal is provided by an arbiter."

176. AMD documentation as of August 24, 2001, described a shader and arbiter having the same characteristics as the shader and arbiter of the '454 Patent. As stated previously, the sequencer implements its arbitration scheme by generating control signals, for both pixel and vertex operations and instructions, in the form of control packets for either vector.

177. For vertex operations, the control packets contain the state pointer, the tag to the position cache and a register file base pointer. The control packets are sent to the reservation stations within the unified shader. From these control packets, the instructions for clauses are fetched from the global instruction store. The control packet is then sent to a subsequent reservation station. All the instructions are received and the control packet is sent to the next reservation station. The control packet continues to travel down the path of reservation stations until all clauses have been executed.

13.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 16 vertices (actually vertex indices – 32 bits/index for 512 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
   - the 16 vertex indices are sent to the 16 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of texture clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
      - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
      - parameter data is sent to the Parameter Cache over a dedicated bus
      - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
      - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

See EX. 2010 (R400 Sequencer Spec), at pp. 17-18.

178. For pixel operations, the sequencer control starts with interpolation of

parameters by sending barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator. The interpolator generates parameter values for each pixel. The sequencer control transfers the interpolated data to the SP register file over the RE_SP interface. The sequencer then constructs a control packet for the vector and sends it to the first reservation station. The control packet is then sent to the next reservation station. The control packet continues to travel down the reservation station path until all clauses have been executed.

13.1.2 *Sequencer Control of a Vector of Pixels*

1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector
   - after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer
   - the Parameter buffer is wide enough to source 3 vertices worth of a particular parameter in one cycle
   - at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA
3. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIFO
   - the Pixel FIFO is wide enough to source one quad's worth of barycentrics per cycle
4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
5. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
6. SEQ control starts with the interpolation of parameters (up to 16 per thread) by sending the barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator
   - P0i, P0j, and P0k (the value of P0 at each vertex) are loaded into the interpolator from the Parameter buffer
   - Q0 i, j, and k are loaded into the interpolator from the Pixel FIFO
   - The interpolator then generates the parameter value for each pixel in Q0 (Q0P0)
   - P0i, P0j, and P0k are sent to the interpolator for Q1 only if Q1 is from a different primitive; if Q1 is from the same primitive as Q0, then the P0i, P0j, and P0k values loaded for Q0 are held by the interpolator and reused for Q1
     - a "different_prim" control bit is passed with the barycentric data for each quad in the Pixel FIFO that indicates whether new parameter data needs to be loaded into the interpolator
   - Q1 i, j, and k are then loaded into the interpolator from the Pixel FIFO
   - The interpolator then generates the parameter value for each pixel in Q1 (Q1P0)
   - Q2P0 and Q3P0 are generated in a similar manner
   - The next set of parameter data - P1i, P1j, and P1k - is then loaded into the interpolator
   - Q0 i, j, and k now must be re-read from the Pixel FIFO – this means that the output of the Pixel FIFO loops through the top four entries on each read command until at the end a final "block_pop" signal is asserted, causing the top four sets of barycentric coordinates to finally be removed
   - so the order of parameter info generated is Q0P0, Q1P0, Q2P0, Q3P0, Q0P1, Q1P1, etc.
7. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
   - 16 pixels worth of interpolated parameter data is sent to the 16 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written with Q0P0 the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written with Q1P0 second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written with Q2P0 third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written with Q3P0 fourth cycle

8. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other informations (such as quad address for example) travels in a separate FIFO
9. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
10. all instructions of texture clause 0 are issued by TSM0
11. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for texture requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
12. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
13. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
14. the control packet continues to travel down the path of reservation stations until all clauses have been executed
   - pixel data is exported in the last ALU clause (clause 7)
     - it is sent to an output FIFO where it will be picked up by the render backend
     - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
15. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

See EX. 2010 (R400 Sequencer Spec), at pp. 18-19.

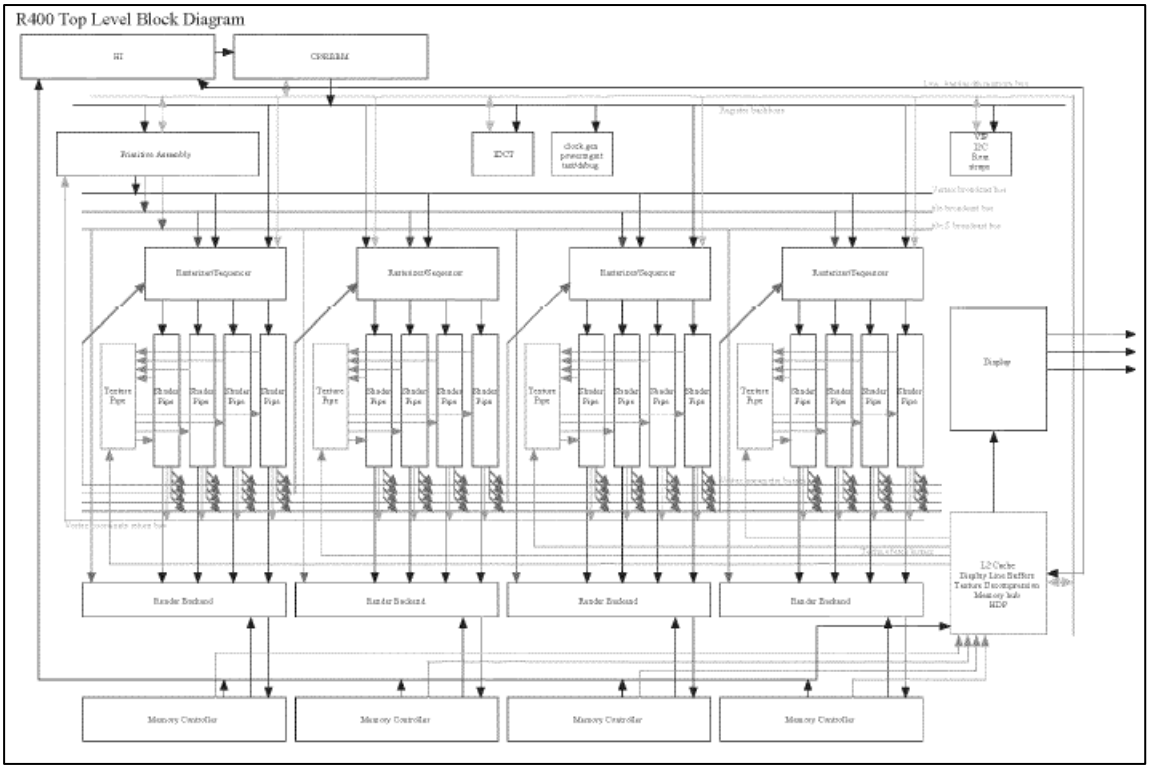
11. Claim 11

a. "A unified shader comprising:"

179. The preamble of Claim 11 recites "A unified, comprising."

180. It is my opinion that the R400 is a unified shader.

181. AMD documentation as of August 24, 2001, described a unified shader having the same characteristics as the unified shader of the '454 Patent. For example, the Top Level Specification states that the R400 is a "standalone graphics chip product." *See* EX. 2013 (R400 Top Level Spec), at p. 6. Thus, the R400 is a graphics processor. Further, the Top Level Specification describes the R400 as having the following functional blocks including a shader processor and sequencer.



7. Blocks

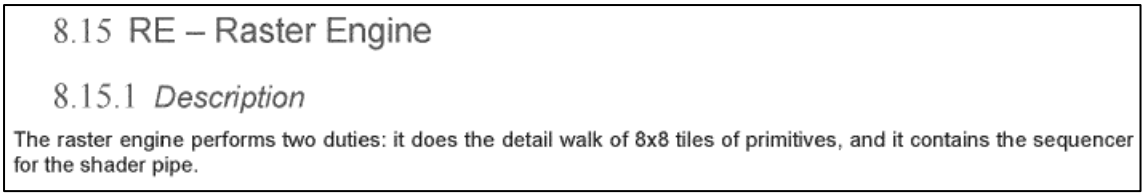
HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller

See EX. 2013 (R400 Top Level Spec), at pp. 15-16.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

See Id at p. 11.



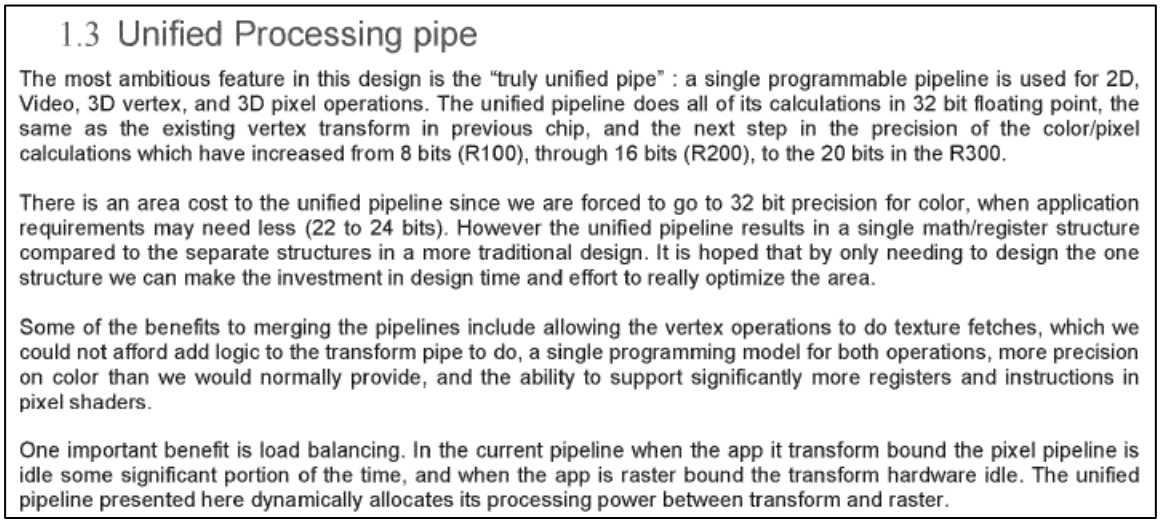
8.15 RE – Raster Engine

8.15.1 *Description*

The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

See Id. at p. 27.

182. Moreover, the R400 implements a unified processing pipe to perform 3D vertex and pixel operations.



1.3 Unified Processing pipe

The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

183. Specifically, the R400 utilizes a unified shader, which comprises a SIMD/vector engine that performs pixel shader operations and vertex shader operations, to render 3D graphics. *See* EX. 2013 (R400 Top Level Spec), at pp. 9-10. The unified processing pipeline of the R400, including its unified shader, performs 3D rendering by transforming vertices and determining the color of pixels. *See Id.* at pp. 10-12.

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

184. The first element of Claim 11 recites "a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload."

185. AMD documentation as of August 24, 2001, described a processor unit having the same characteristics as the processor unit of the '454 Patent. For example, the unified shader implements dynamic load balancing between transform and raster.

> One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See Id. at p. 7.

186. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel calculation operations until enough shared resources become available, at which point the unified shader will perform vertex calculation operations.

> 2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
> - at this point the vector is removed from the Vertex FIFO
> - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 17.

> 4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 18.

> The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

187. As a result, the ALU clauses are executed in an interleaved manner to hide the ALU latency.

> The sequencer is based on the R300 design. It chooses an two ALU clauses and a texture clause to execute, and executes all of the instructions in aa clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the top bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

See Id. at p. 4.

188. Accordingly, there will always be two active ALU clauses at any given time.

> There will always be two active ALU clauses at any given time (and two arbitrers) In this case, the instructions of a vector are interleaved with the instructions of the other vector. One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

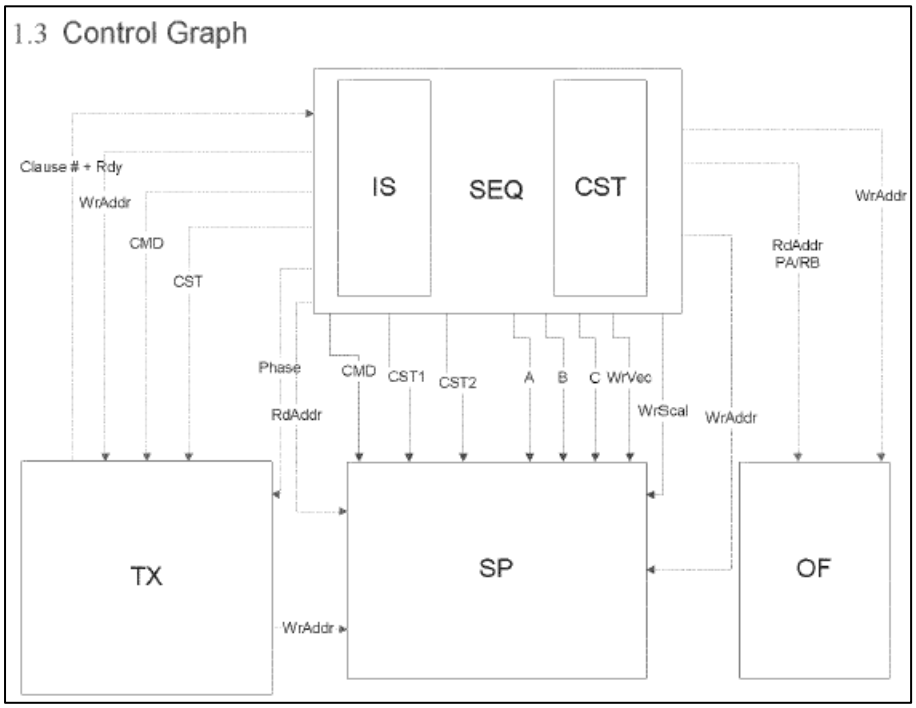
See Id. at p. 6.

c. "an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store."

189. The final element of Claim 11 recites "an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations

and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store."

190. AMD documentation as of August 24, 2001, described an instruction store having the same characteristics as the instruction store of the '454 Patent. For example, as stated previously, the R400 includes an instruction store.



See Id. at p. 12.

191. The processor unit of the unified shader performs vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the store. For example, the unified shader implements dynamic load balancing between transform and raster.

> One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

See EX. 2013 (R400 Top Level Spec), at p. 7.

192. Based on the arbitration scheme implemented by the sequencer, the unified shader will perform pixel manipulation operations until enough shared resources become available, at which point the unified shader will perform vertex manipulation operations.

> 2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
>   - at this point the vector is removed from the Vertex FIFO
>   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

See EX. 2010 (R400 Sequencer Spec), at p. 17.

> 4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

See Id. at p. 18.

> The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32~~-~~16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

See Id. at p. 5.

193. As a result, the ALU clauses are executed in an interleaved manner to hide the ALU latency.

> The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in aa clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

See Id. at p. 4.

194. Accordingly, there will always be two active ALU clauses at any given time.

> ~~There will always be two active ALU clauses at any given time (and two arbitrers) In this case, the instructions of a vector are interleaved with the instructions of the other vector.~~ One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

See Id. at p. 6.

IX. DILIGENCE IN REDUCING TO PRACTICE

195. The '454 Patent is directed to graphics processing architectures employing a unified shader. *See generally* EX. 1001. The named inventors of patents—Stephen Morein, Andrew Gruber, Andi Skende and Laurent Lefebvre—all worked at ATI and collectively conceived of the claimed invention while designing a graphics-processing unit ("GPU") known as the R400. *Id.*; *see also* EX. 2102 (Lefebvre Dep.), at 76:20-78:25. The inventors conceived these inventions no later than August 24, 2001. Evidence of this is contained in written technical specifications describing the R400 GPU. *See* Section VIII, *supra*.

196. These inventors, aided by a team of engineers, created an implementation of the R400 in source code, *e.g.*, emulation code and RTL. A team of engineers worked every non-holiday workday towards reducing the '454 Patent to practice. *See* EX. 2004 (Lefebvre Decl., IPR2015-00325), at pp. 23, 33-58 (diligence calendar showing work on R400 documents from August 2001 to November 2003); EX. 2102 (Lefebvre Dep.) at 257:4-11; EX. 2113 (Gruber Dep.),

at 202:2-9; *see generally* EX. 2007 (IPR2015-0325, 326, 330 Lefebvre Dep.); EX. 2118; EX. 2119. Engineers involved in the development of the R400 include numerous ATI employees identified in the project allocation outlines of the R400 Program Reviews. EX 2111 (R400 Program Reviews) at pp. 1-6, 16-52; EX. 2015 (Microsoft Site Visit), at p. 4; *see generally* EX. 2110 (R400 Program Reviews); EX. 2119. In the beginning of development, only a handful of engineers were assigned to the R400 project. By late 2001, ATI assigned over one hundred project managers/designers to implement and test the R400. *See* EX. 2113 (Gruber Dep.) at 201:22-202:9. These project managers/designers diligently worked on the R400. *See id.* They used the specifications to write emulation code and RTL code for the R400's functional blocks. They then tested the R400's functional blocks.

197. Generally, GPUs spend a long time in development; they take "extensive time to build." *See* EX. 2102 (Lefebvre Dep.), at 56:3-8, 97:7-98:6; *see generally* EX. 2006 (IPR2015-00325, 326, 330 Wolfe Dep.).

198. The R400 team at ATI continued to develop an implementation of the R400 that was synthesized to a netlist and downloaded onto a field programmable gate array (FPGA). The inventors constructively reduced their inventions to practice on November 20, 2003 by way of filing the patent applications that led to the grant of the '454 Patent. ATI diligently reduced to practice the inventions of the '454 Patent throughout the period of August 24, 2001 to November 20, 2003. This work

is shown by the following: (i) continuous updates to the R400 Technical specifications; (ii) development, testing, and debugging of emulation code and RTL code for the R400, including all critical components, and (iii) synthesis of netlists to be placed on the IKOS system for testing. *See* EX. 2004 (Lefebvre Decl., IPR2015-00325), at pp. 21-27; 33-60; *see generally* EXS. 2110-2112, EX. 2103, EXS. 2104-2109; EX. 2118; EX. 2119.

199. Development of the R400 included the completion of certain deliverables. The deliverables are indications of the conception and reduction to practice of the claimed elements of the '454 Patent. The design cycle involves completing and producing the following deliverables: (i) specification drafting; (ii) software emulation code development with functional verification plans and tests; (iii) RTL code development with functional verification plans and tests; (iv) netlist synthesis; and (v) IKOS testing. *See generally* EX. 2103.

200. The technical specifications are documents outlining the functional features and architecture of the components comprising the R400. Software emulation code (C++) emulates the behavior of the graphics-processing system using software that executes on a computer. *See* EX. 2002 (Declaration of Andrew Wolfe, IPR2015-00325), at pp. 14-18; *see also generally* EX. 2006 (IPR2015-00325, 326, 330 Wolfe Dep.). RTL is a hardware-description language that is used to design and verify digital circuits at register-transfer level of abstraction that can

include both structure and function. *Id.* A netlist is synthesized from the RTL and is a description of the connectivity of an electronic circuit. A netlist is a layout file that is used to fabricate a chip. *Id.* The IKOS system is a plurality of field programmable gate arrays (FPGAs) that can be configured with a netlist synthesized from RTL. In the development of the R400, numerous netlists were synthesized from milestone versions of RTL that had achieved functional completeness. These netlists were placed on the IKOS hardware for further testing.

201. During development, the architectural leads for each block wrote specifications to describe the structure and function of the blocks. *See generally*, *e.g.*, EX. 2008; EX. 2011; EX. 2037; EXS. 2066-2067; EX. 2104; EXS. 2105 to 2109. Specifically, Laurent Lefebvre updated the R400 Sequencer Specification approximately every two to three weeks. *See* EX. 2104; EXS. 2105 to 2109 (outlining edits to the document). There are at least thirty-three revisions of this document. *See id*. The revisions span from at least May 25, 2001, to August 29, 2003. *See id*.

202. Having multiple revisions during development is typical. Each revision showed progress during the previous week(s). During interim periods, the project team worked on outstanding issues. *See*, *e.g.*, EX. 2106, at p. 35 (ending the document with a section labeled "Open Issues").

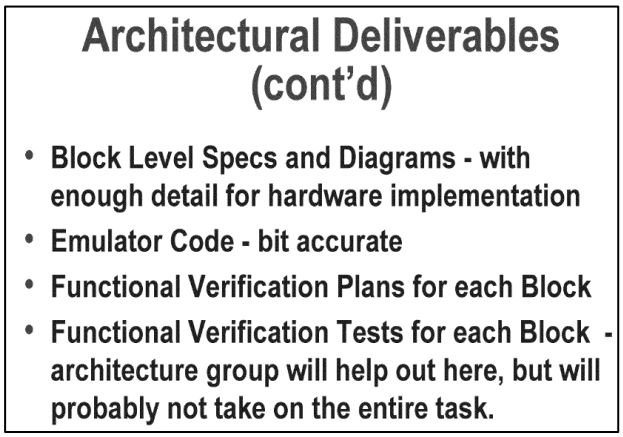
203. Everyone assigned to the R400 project saved their work in a revision

control system, called Perforce. Perforce is a revision-controlled repository that maintains time stamped logs, user permissions, and whether files are checked out. *See* EX. 2003 (Watson Decl., EX. 2105 325 IPR), at pp. 10-11; *see generally* EX. 2005 (IPR2015-0325, 326, 330 Watson Dep.). Accordingly, Perforce captured every change made to files describing the design and development of the R400 product. *See* EX. 2009; EXS. 2029; EX. 2030; EX. 2031; EX. 2035; EX. 2036; EX. 2068; EXS. 2110-2111; *see also* EX. 2102 (Lefebvre Dep.), at 208:12-201:13, 219:19-221:20.

204. Perforce stores numerous presentations, technical specifications, testing code, source code, change lists, and logs related to the development of the R400 on the IKOS FPGAs. *See* EX. 2102 (6/28/2017 Lefebvre Dep.), at 208:12-210:13, 219:19-221:20.

205. Throughout the development of the R400, the team of engineers working on the project tracked their efforts in program review presentations. The presentations cataloged the efforts of the team, at least, during the years of 2001-2003. The program reviews included information related to the development of all development deliverables for the R400 project including technical specification drafting, emulation code development for each block in the R400, RTL code development for each block in the R400, verification and validation of all emulation and RTL features, netlist synthesis, IKOS testing, hardware development, software

development, physical design layout, and many other topics. *See* EX. 2014; EXS. 2016-2023 EX. 2034; EXS 2039-2043; EX 2044-2055; EX 2056-2065; EX. 2070; EX. 2119. A program review listing the first set of deliverables authored during the development of the R400 is shown below.



EX. 2111 at pp. 7, 11 (R400 Program Review, 12/13/2001); *see generally* EXS. 2110-2111.

Milestone	Plan
Top Level	
MPEG Plan	12/17
Block Diagram	Done
Suggested Floor Plan	Done
2D Programming guide	12/19
Shader Guide	12/19
Programming Guide	12/24
Area Estimate	2/1
Synchronization and Coherency Document	1/15
HOS Document	1/15
Multi Media Guide	2/1
Performance Verification Plan	2/1

Milestone	Plan
Block Level Specs complete	2/15
Block Level Emulators Phase 1	1/15
Block Level Emulators Phase 2	3/1
Block Level Emulators Phase 3	4/15
Block Verification Plans Complete	4/1

EX. 2111 at pp. 7, 14-15 (R400 Program Review, 12/13/2001); *see generally* EXS. 2110-2111.

206. Following the specification and emulation tasks, RTL feature sets were developed. Consistent with ATI's development practice, an RTL feature complete target date was set once emulation and specification drafting was complete.

8/1/02
- **All blocks submitted to GC w/ 80% functionality (enough for most perf testing)**

9/15/02
- **All blocks submitted to GC, RTL feature complete**

11/15/02
- **RTL freeze (debug complete)**

EX. 2111 (R400 Program Review, 12/13/2001), at pp. 32-52; *see generally* EXS. 2110-2111.

207. The following program schedule captures the critical milestones to be completed throughout the development of the R400. Notably, the program schedule includes emulation completion, Verilog feature completion, netlist synthesis, and

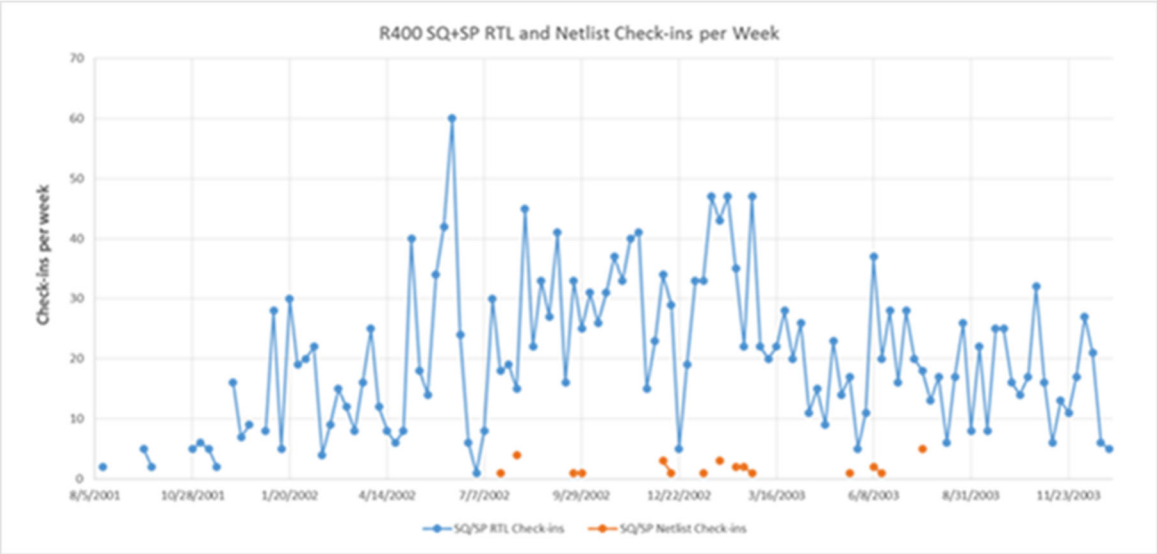
IKOS testing.

Task	Plan	Actual	Forecast
Significant Architecture Issue Identification Complete	10-15-01	10-10-01	
Significant Architectural Issues Resolved	12-17-01	12-20-01	
Emulator Test template Complete	01-18-02		01-18-02
PrimLib has calls for basic tests	02-01-02		02-01-02
Updated Area Estimate Complete / Specs Complete	02-15-02		02-15-02
GC Emulator integration – 1 triangle	02-22-02		02-22-02
Core Emulator pixel / shader tests	03-15-02		03-15-02
Block Testing Begins	04-16-02		04-16-02
GC/Chip Integration Start	05-17-02		05-17-02
Simulate 1 Triangle / Emulator ready for SW	06-03-02		06-03-02
First Syntheses	07-12-02		07-12-02
Verilog Feature Complete	09-16-02		09-16-02
IKOS Emulation start	10-11-02		10-11-02
Synthesis meets Timing	10-25-02		10-25-02
Verilog Frozen	11-08-02		11-08-02
IKOS Emulation (w/ Software) begins	11-11-02		11-11-02
Final Netlist (Gate level ECO only)	11-15-02		11-15-02
A11 Base Layers Tapeout	01-10-03		01-10-03
A11 Metal Layers Tapeout	01-24-03		01-24-03
First Samples for engineering evaluation	03-12-03		03-12-03
A12, A13 Netlist			
A12, A13 Samples			
Volume Ramp			

EX. 2111 (R400 Program Review, 1/17/2002), at pp. 53-69; *see generally* EXS. 2110-2111.

208. Perforce also maintained all changes made to the RTL block level development files. *See* EX. 2003 (Watson Decl., EX. 2105 325 IPR), at pp. 66; EX. 2004 at ¶ 39; EX. 2007 at 157:15-158:7; *see generally* EX. 2005 (IPR2015-0325, 326, 330 Watson Dep.). This information is stored in the form of changelists and file logs. These files identify the date a change was made, who made the change, and what change was made. *Id.* These files show that diligence was utilized during development of the SQ, SP, PA, RB, SC, SPI, SX, VC, and VGT blocks incorporated in the R400 from August 24, 2001 (the date of conception), to November 20, 2003 (the date of the filing of the patent application to the '454 patent). Work on certain

of these blocks began before other blocks. The chart below details activity on two of these blocks – SP and SQ. The evidence underlying the chart is described in the block changelists and file logs relating to changes to the SP and SQ blocks of the R400 RTL and netlists from August 5, 2001 through the end of November 2003, and is further discussed and cited to in paragraphs 209-212 below. The PA, RB, SC, SPI, SX, VC, and VGT blocks are further discussed in paragraphs 213-236 below. The frequency of check-ins for the modules demonstrates that the R400 design team exercised diligence during the reduction to practice of the R400.



209. Edits to the SQ, or shader sequencer, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. Edits to the SQ block, as demonstrated by changelists and file logs, began in March 2002 and continued through November 19, 2003. At a high level, the SQ block maintains instruction buffers, and is responsible for coordinating execution of

instructions. See EX. 2079, at p. 6. Engineers began editing the RTL code associated with the SQ block in November 2001. Starting in March 2002 and continuing through November 18, 2003, multiple engineers made changes to the RTL files on a daily or weekly basis. These RTL files are part of the SQ block, and allow the inventions' sequencer to cause the appropriate operands to be transferred from the general purpose register block into the appropriate source registers for execution. *See* EX. 2112, at R400 File Logs, 8/24/2001-9/30/2002, sq.v.log, R400 File Logs, 4/18/2003-11/20/2003, sq.v.log; *see* EX. 2098 at R400 File Logs, 10/1/2002 - 4/17/2003, sq.v.log.

210. Initial revisions of the SQ block began in November 2001 and continued through 2002. By March 2002, and continuing through November 19, 2003, the SQ block was being checked-out on an approximately daily or weekly basis by multiple individuals as ATI worked to implement its design. *See* EX. 2112 at Block Changelist 8/24/2001-9/30/2002, SQ_check-in_history.txt; Block Changelist 4/18/2003 - 11/20/2003, SQ_check-in_history.txt, *see* EX. 2095, Block Changelist 10/1/2002 - 4/17/2003, SQ_check-in_history.txt.

211. Edits to the SP, or shader pipe, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. Edits to the SP block, as demonstrated by changelists and file logs, began in September 2001 and continued through November 19, 2003. At a high level, the SP

block implements the math pipeline of the R400. See EX. 2028, at p. 30. Engineers began editing the RTL code associated with the SP block in November 2001. Starting in February 2002 and continuing through November 19, 2003, the engineers made changes to the RTL files on a daily or weekly basis. These RTL files are part of the SP block and are related to the unified shader of the inventions. See EX. 2112, at R400 File Logs, 8/24/2001-9/30/2002, sp.v.log, R400 File Logs, 4/18/2003-11/20/2003, sp.v.log; see EX. 2098, at R400 File Logs, 10/1/2002 - 4/17/2003, sp.v.log.

212. Initial revisions of the SP block began in September 2001. Beginning in November 2001 and continuing through November 19, 2003, changes were made to the SP block on a daily or weekly basis. During this period, the SP block was being checked-out by multiple individuals on an approximately daily or weekly basis as ATI worked on functionally implementing its design. See EX 2112, at Block Changelist 8/24/2001-9/30/2002, SP_check-in_history.txt, Block Changelist 10/1/2002 - 4/17/2003, SP_check-in_history.txt, Block Changelist 4/18/2003 - 11/20/2003, SP_check-in_history.txt.

213. Edits to the PA, or primitive assembly, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. Edits to the PA block, as demonstrated by changelists and file logs, began in March 2002 and continued through October 9, 2003. At a high level, the PA block

performs vertex processing, primitive processing, rasterizes primitives into pixels, and performs pixel processing, and as such is a central component of a unified shader. *See* EX. 2038, at p. 4. Engineers began editing the RTL code associated with the primitive assembly block in March 2002, and continued editing the RTL files on a daily or weekly basis through January 13, 2003, after which time periodic revisions were made to the RTL files through October 9, 2003. *See* EX. 2112 (R400 File Logs, 8/24/2001-9/30/2002, pa.v.log), at p. 406; *see* EX. 2098 (R400 File Logs, 10/1/2002 - 4/17/2003, pa.v.log); *see* EX. 2112, (R400 File Logs, 4/18/2003-11/20/2003, pa.v.log), at p. 113. These RTL files are part of the PA block, and are tied to the unified shader of the claimed inventions.

214. Initial revisions of the PA block began in September 2001 and continued on a near-daily basis through the end of the September, with monthly updates being performed through January 2002. By January 2002, multiple developers worked on the PA block, as demonstrated by daily and weekly changes to the PA block code, continuing through November 19, 2003. During this period, the PA block was being checked-out by multiple individuals on an approximately daily basis as ATI worked on implementing its design. *See* EX. 2112, (Block Changelist 8/24/2001-9/30/2002, PA_check-in_history.txt), at, p. 590; *Id.* (Block Changelist 10/1/2002-4/17/2003, PA_check-in_history.txt), at, p. 583; *Id.* (Block Changelist 4/18/2003-11/20/2003), at p. 265.

215. Edits to the RB, or render backend, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. Edits to the RB block, as demonstrated by changelists and file logs, began in December 2001 and continued through November 13, 2003. At a high level, the RB block performs all of the fixed-function operations on depth and color values, compresses and decompresses depth and color data, and performs shadow buffer comparisons and anti-aliased filtering for the texture pipe. *See* EX. 2028, at p. 32. The RTL files associated with the RB block allow the receipt of vertex data generated by the processor, and convert the vertex data into one of several formats suitable for later display on a device.

216. Initial revisions of the RB block began in December 2001. Starting in March 2002, multiple developers performed daily and weekly changes to the RB block code. RB block files show that the RB block was being checked-out by multiple individuals on a daily and weekly basis through November 13, 2003, as ATI made changes to implement its design. *See* EX. 2112, at Block Changelist 8/24/2001-9/30/2002, RB_check-in_history.txt, Block Changelist 10/1/2002 - 4/17/2003, RB_check-in_history.txt; Block Changelist 4/18/2003 - 11/20/2003, RB_check-in_history.txt.

217. Edits to the SC, or scan converter, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003.

Edits to the SC block, as demonstrated by changelists and file logs, began in March 2002 and continued through November 19, 2003. Engineers began editing the RTL code associated with the SC block in March 2002, and continued making changes to the RTL files on a daily or weekly basis through December 31, 2002. After this time, changes were periodically made to the RTL files through November 19, 2003. These RTL files are part of the SC block, and relate to the conversion of data into a format suitable for display on a device. *See* EX. 2112, at R400 File Logs, 8/24/2001-9/30/2002, sc.v.log, R400 File Logs, 4/18/2003-11/20/2003, sc.v.log); EX. 2098 at R400 File Logs, 10/1/2002 - 4/17/2003, sc.v.log).

218. Initial revisions of the SC block began in March 2002 and continued on a near-daily basis through November 19, 2003. During this period, the SC block was being checked-out by multiple individuals on an approximately daily basis as ATI worked to functionally implement its design. See EX. 2112, at Block Changelist 8/24/2001-9/30/2002, SC_check-in_history.txt, Block Changelist 4/18/2003 - 11/20/2003, SC_check-in_history.txt; see EX. 2092, Block Changelist 10/1/2002 - 4/17/2003, SC_check-in_history.txt.

219. Edits to the SPI, or shader processor input, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. Edits to the SPI block, as demonstrated by changelists and file logs, began in April 2003 and continued through November 6, 2003. At a high level, the SPI

block manages shader resources and provides shader input data to the general purpose register of the present invention. See EX. 2027, at p. 6.

220. Initial revisions of the SPI block began in April 2003, after which the SPI block was being checked-out by multiple individuals on an approximately daily or weekly basis through November 6, 2003, as ATI worked on implementing its design. See EX. 2093, Block Changelist 10/1/2002 - 4/17/2003, SPI_check-in_history.txt, Block Changelist 4/18/2003 - 11/20/2003, SPI_check-in_history.txt.

221. Edits to the SX, or shader export, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. Edits to the SX block, as demonstrated by changelists and file logs, began in November 2001 and continued through November 19, 2003. At a high level, the SX block is responsible to receive and re-order color and position exports from the shader core and forward them to the correct client. See EX. 2025, at p. 5. Engineers began editing the RTL code associated with the SX block in December 2001. Starting in March 2002 and continuing through November 6, 2003, multiple engineers made changes to the RTL files on a daily or weekly basis (with a one month pause in October 2002). These RTL files are part of the SX block, and allow the shader to export data once processed. *See* EX. 2112 at R400 File Logs, 8/24/2001-9/30/2002, sx.v.log, R400 File Logs, 4/18/2003-11/20/2003, sx.v.log; *see* EX. 2098 at R400 File Logs, 10/1/2002 - 4/17/2003, sx.v.log.

222. Initial revisions of the SX block began in November 2001 and continued through 2002. By March 2002, and continuing through November 19, 2003, the SX block was being checked-out on an approximately daily or weekly basis by multiple individuals as ATI worked to implement its design. *See* EX. 2112 at Block Changelist 8/24/2001-9/30/2002, SX_check-in_history.txt, Block Changelist 4/18/2003 - 11/20/2003, SX_check-in_history.txt; *see* EX. 2096 (Block Changelist 10/1/2002 - 4/17/2003, SX_check-in_history.txt).

223. Edits to the VC, or vertex cache, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. At a high level, the vertex cache receives and stores vertex information from the vertex block of the inventions. Initial revisions of the VC block began in May 2003, and the VC block was being checked-out by multiple individuals on an approximately daily or weekly basis through November 14, 2003, as ATI worked on implementing its design. *See* EX. 2112 at Block Changelist 4/18/2003 - 11/20/2003, VC_check-in_history.txt.

224. Edits to the VGT, or vertex geometry tesselator, block help demonstrate that diligence in the R400 began on August 24, 2001 and continued through November 20, 2003. At a high level, the VGT block is the primary block responsible for supporting all DX and OpenGL draw packets. *See* EX. 2026, at p. 8.

225. Initial revisions of the VGT block began in January 2002, and

continued on a bi-weekly basis until March 2002. Starting in March 2002, and continuing through November 6, 2003, the VGT block was being checked-out by multiple individuals on an approximately daily or weekly basis as ATI worked to implement its design. *See* EX. 2112 (Block Changelist 8/24/2001-9/30/2002, VGT_check-in_history.txt), at pp. 564-589; *see* EX. 2097 (Block Changelist 10/1/2002 - 4/17/2003, VGT_check-in_history.txt); *see* EX 2112 (Block Changelist 4/18/2003 - 11/20/2003, VGT_check-in_history.txt), at pp. 260-264.

226. The work on the PA, RB, SC, SP, SPI, SQ, SX, VC, and VGT blocks of the R400 show that that the inventors exercised diligence in reducing to practice the inventions of the '454 patent.

Block Changelist 8/24/2001 – 9/30/2002:

EX. 2112:

- Block Changelist 8/24/2001-9/30/2002, RB_check-in_history.txt, at pp. 419-466;
- Block Changelist 8/24/2001-9/30/2002, SX_check-in_history.txt, at pp. 467-476;
- Block Changelist 8/24/2001-9/30/2002, SP_check-in_history.txt, at pp. 477-491;
- Block Changelist 8/24/2001-9/30/2002, SQ_check-in_history.txt, at pp. 492-517;
- Block Changelist 8/24/2001-9/30/2002, SC_check-in_history.txt, at pp. 518-563;
- Block Changelist 8/24/2001-9/30/2002, VGT_check-in_history.txt, at pp. 564-589;
- Block Changelist 8/24/2001-9/30/2002, PA_check-in_history.txt, at pp. 590-683;

Block Changelist 10/1/2002 – 4/17/2003:

- EX. 2090, Block Changelist 10/1/2002 - 4/17/2003, PA_check-in_history.txt;
- EX. 2091, Block Changelist 10/1/2002 - 4/17/2003, RB_check-in_history.txt;
- EX 2092, Block Changelist 10/1/2002 - 4/17/2003, SC_check-in_history.txt;
- EX. 2093, Block Changelist 10/1/2002 - 4/17/2003, SPI_check-in_history.txt;
- EX. 2094, Block Changelist 10/1/2002 - 4/17/2003, SP_check-in_history.txt;
- EX. 2095, Block Changelist 10/1/2002 - 4/17/2003, SQ_check-in_history.txt;
- EX. 2096, Block Changelist 10/1/2002 - 4/17/2003, SX_check-in_history.txt;
- EX. 2097, Block Changelist 10/1/2002 - 4/17/2003, VGT_check-in_history.txt.

Block Changelist 4/18/2003 – 11/20/2003:

- EX. 2097, Block Changelist 4/18/2003 - 11/20/2003, VGT_check-in_history.txt.

EX. 2112:

- Block Changelist 4/18/2003 - 11/20/2003, RB_check-in_history.txt, at pp. 118-151;
- Block Changelist 4/18/2003 - 11/20/2003, SPI_check-in_history.txt, at pp. 152-157;
- Block Changelist 4/18/2003 - 11/20/2003, SX_check-in_history.txt, at pp. 158-176;
- Block Changelist 4/18/2003 - 11/20/2003, SP_check-in_history.txt, at pp. 177-195;
- Block Changelist 4/18/2003 - 11/20/2003, SQ_check-in_history.txt, at pp. 196-248;
- Block Changelist 4/18/2003 - 11/20/2003, SC_check-in_history.txt, at pp. 249-259;
- Block Changelist 4/18/2003 - 11/20/2003, VGT_check-in_history.txt, at pp. 260-264;
- Block Changelist 4/18/2003 - 11/20/2003, PA_check-in_history.txt, at pp. 265-271;
- Block Changelist 4/18/2003 - 11/20/2003, VC_check-in_history.txt, at pp. 272-318;

R400 File Logs 8/24/2001 – 9/30/2002:

EX. 2112:

- R400 File Logs, 8/24/2001-9/30/2002, sq_alu_instr_seq.v.log, at pp. 319-320;

- R400 File Logs, 8/24/2001-9/30/2002, sq.v.log, at pp. 321-331;
- R400 File Logs, 8/24/2001-9/30/2002, sq_alu_instr_queue.v.log, at pp. 332-333;
- R400 File Logs, 8/24/2001-9/30/2002, sq_tex_instr_seq.v.log, at pp. 334-335;
- R400 File Logs, 8/24/2001-9/30/2002, sq_vtx_thread_buff.v.log, at p. 336;
- R400 File Logs, 8/24/2001-9/30/2002, tp.tree.log, at pp. 337-340;
- R400 File Logs, 8/24/2001-9/30/2002, sq_ctl_flow_seq.v.log, at pp. 341-345;
- R400 File Logs, 8/24/2001-9/30/2002, tp.v.log, at pp. 346-347;
- R400 File Logs, 8/24/2001-9/30/2002, sq_input_arb.v.log, at p. 348;
- R400 File Logs, 8/24/2001-9/30/2002, sx_export_buffers.v.log, at p. 349;
- R400 File Logs, 8/24/2001-9/30/2002, sq_tex_instr_queue.v.log, at p. 350;
- R400 File Logs, 8/24/2001-9/30/2002, sq_defs.v.log, at pp. 351-352;
- R400 File Logs, 8/24/2001-9/30/2002, sq_instruction_store.v.log, at pp. 353-354; A
- R400 File Logs, 8/24/2001-9/30/2002, sp.v.log, at pp. 355-359;
- R400 File Logs, 8/24/2001-9/30/2002, sq_target_instr_fetch.v.log, at pp. 360-361;
- R400 File Logs, 8/24/2001-9/30/2002, sp_macc32.mc.log, at pp. 362-363;
- R400 File Logs, 8/24/2001-9/30/2002, sp_macc_gpr.v.log, at pp. 364-367;
- R400 File Logs, 8/24/2001-9/30/2002, pa_ccg_sxifsm.v.log, at pp. 368-369;
- R400 File Logs, 8/24/2001-9/30/2002, sx.v.log, at pp. 370-374
- R400 File Logs, 8/24/2001-9/30/2002, sp_macc.v.log, at pp. 375-379
- R400 File Logs, 8/24/2001-9/30/2002, sc.v.log, at pp. 379-387;
- R400 File Logs, 8/24/2001-9/30/2002, sq_ais_output.v.log, at pp. 388-390;
- R400 File Logs, 8/24/2001-9/30/2002, sq_thread_arb.v.log, at pp. 391-393;
- R400 File Logs, 8/24/2001-9/30/2002, sp_vector.v.log, at pp. 394-399;
- R400 File Logs, 8/24/2001-9/30/2002, sq_pix_thread_buff.v.log, at p. 400;
- R400 File Logs, 8/24/2001-9/30/2002, sx_parameter_caches.v.log, at p. 401;
- R400 File Logs, 8/24/2001-9/30/2002, sx_export_control.v.log, at p. 402;
- R400 File Logs, 8/24/2001-9/30/2002, sq_export_alloc.v.log, at p. 403-404;
- R400 File Logs, 8/24/2001-9/30/2002, sx_param_cache_ctl.v.log, at p. 405;
- R400 File Logs, 8/24/2001-9/30/2002, pa.v.log, at pp. 406-414
- R400 File Logs, 8/24/2001-9/30/2002, pa_ag.v.log, at pp. 415-416;
- R400 File Logs, 8/24/2001-9/30/2002, pa_sxifccg.v.log, at pp. 417-418;

R400 File Logs 10/1/2002 – 4/17/2003:

EX. 2098 at:

ATI Ex. No 2121
IPR2023-00922
Page 138 of 324

- R400 File Logs, 10/1/2002 - 4/17/2003, pa.v.log, at pp. 1-7;
- R400 File Logs, 10/1/2002 - 4/17/2003, tp.tree.log, at pp. 8-9;
- R400 File Logs, 10/1/2002 - 4/17/2003, pa_ag.v.log, at p. 10;
- R400 File Logs, 10/1/2002 - 4/17/2003, pa_ccg_sxifsm.v.log, at pp. 11-12;
- R400 File Logs, 10/1/2002 - 4/17/2003, pa_sxifccg.v.log, at pp. 13-17;
- R400 File Logs, 10/1/2002 - 4/17/2003, sc.v.log, at pp. 18-26;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_tex_instr_queue.v.log, at pp. 27-28;
- R400 File Logs, 10/1/2002 - 4/17/2003, sp.v.log, at pp. 29-31;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_tex_instr_seq.v.log, at pp 32-33;
- R400 File Logs, 10/1/2002 - 4/17/2003, sp_macc.v.log, at pp. 34-37;
- R400 File Logs, 10/1/2002 - 4/17/2003, sp_macc32.mc.log, at pp. 38-44;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_thread_arb.v.log, at p. 45;
- R400 File Logs, 10/1/2002 - 4/17/2003, sp_macc_gpr.v.log, at pp. 46-48;
- R400 File Logs, 10/1/2002 - 4/17/2003, sp_vector.v.log, at pp. 49-54;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq.v.log, at pp. 55-67;
- R400 File Logs, 10/1/2002 - 4/17/2003, sx_parameter_caches.v.log, at pp. 68-69;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_ais_output.v.log, at pp. 70-75;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_alu_instr_queue.v.log, at pp. 76-79;
- R400 File Logs, 10/1/2002 - 4/17/2003, sx_export_buffers.v.log, at pp. 80-81;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_alu_instr_seq.v.log, at pp. 82-86;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_ctl_flow_seq.v.log, at pp. 87-93;
- R400 File Logs, 10/1/2002 - 4/17/2003, sx_export_control.v.log, at pp. 94-97;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_defs.v.log, at pp. 98-99;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_export_alloc.v.log, at pp. 100-101;
- R400 File Logs, 10/1/2002 - 4/17/2003, sx_param_cache_ctl.v.log, at p. 102
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_input_arb.v.log, at p. 103;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_instruction_store.v.log, at pp. 104-105;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_pix_thread_buff.v.log, at pp. 106-110;
- R400 File Logs, 10/1/2002 - 4/17/2003, sq_target_instr_fetch.v.log, at pp. 111-113;

- R400 File Logs, 10/1/2002 - 4/17/2003, sq_vtx_thread_buff.v.log, at pp. 114-118;
- R400 File Logs, 10/1/2002 - 4/17/2003, sx.v.log, at pp. 119-122.

R400 File Logs 4/18/2003-11/20/2003:

EX 2098 at R400 File Logs, 4/18/2003-11/20/2003, sx.v.log;
EX 2112:

- R400 File Logs, 4/18/2003-11/20/2003, sq_alu_instr_seq.v.log, at pp. 1-7;
- R400 File Logs, 4/18/2003-11/20/2003, sq.v.log, at pp. 8-24;
- R400 File Logs, 4/18/2003-11/20/2003, sq_alu_instr_queue.v.log, at pp. 15-28;
- R400 File Logs, 4/18/2003-11/20/2003, sq_tex_instr_seq.v.log, at pp. 29-32;
- R400 File Logs, 4/18/2003-11/20/2003, sq_vtx_thread_buff.v.log, at pp. 33-40;
- R400 File Logs, 4/18/2003-11/20/2003, tp.tree.log, at pp. 41-42;
- R400 File Logs, 4/18/2003-11/20/2003, sq_ctl_flow_seq.v.log, at pp. 43-48;
- R400 File Logs, 4/18/2003-11/20/2003, sq_input_arb.v.log, at p. 49;
- R400 File Logs, 4/18/2003-11/20/2003, sx_export_buffers.v.log, at p. 50
- R400 File Logs, 4/18/2003-11/20/2003, sq_tex_instr_queue.v.log, at pp. 51-52;
- R400 File Logs, 4/18/2003-11/20/2003, sq_defs.v.log, at pp. 53-55;
- R400 File Logs, 4/18/2003-11/20/2003, sq_instruction_store.v.log, at pp. 56-59;
- R400 File Logs, 4/18/2003-11/20/2003, sp.v.log, at pp. 60-63;
- R400 File Logs, 4/18/2003-11/20/2003, sq_target_instr_fetch.v.log, at pp. 64-66;
- R400 File Logs, 4/18/2003-11/20/2003, sp_macc32.mc.log, at pp. 67-68;
- R400 File Logs, 4/18/2003-11/20/2003, sp_macc_gpr.v.log, at pp. 69-70;
- R400 File Logs, 4/18/2003-11/20/2003, pa_ccg_sxifsm.v.log, at p. 7;
- R400 File Logs, 4/18/2003-11/20/2003, sx.v.log, at pp. 72-76;
- R400 File Logs, 4/18/2003-11/20/2003, sp_macc.v.log, at pp. 77-79;
- R400 File Logs, 4/18/2003-11/20/2003, sc.v.log, at pp. 80-82;
- R400 File Logs, 4/18/2003-11/20/2003, sq_ais_output.v.log, at pp 83-89;
- R400 File Logs, 4/18/2003-11/20/2003, sq_thread_arb.v.log, at pp. 90-92;
- R400 File Logs, 4/18/2003-11/20/2003, sp_vector.v.log, at pp. 93-95;
- R400 File Logs, 4/18/2003-11/20/2003, sq_pix_thread_buff.v.log, at pp. 96-103;

- R400 File Logs, 4/18/2003-11/20/2003, sx_parameter_caches.v.log, at pp. 104-105;
- R400 File Logs, 4/18/2003-11/20/2003, sx_export_control.v.log, at p. 106;
- R400 File Logs, 4/18/2003-11/20/2003, sq_export_alloc.v.log, at pp. 107-111;
- R400 File Logs, 4/18/2003-11/20/2003, sx_param_cache_ctl.v.log, at p. 112;
- R400 File Logs, 4/18/2003-11/20/2003, pa.v.log, at pp. 113-115;
- R400 File Logs, 4/18/2003-11/20/2003, pa_ag.v.log, at p. 116;
- R400 File Logs, 4/18/2003-11/20/2003, pa_sxifccg.v.log, atp., at p. 117;
- EX. 2072, Change Logs.

227. As features of each critical block in the R400 were completed, regression testing was utilized to confirm that iterations of the emulation and RTL code maintained the finalized features. *See* EXS. 2085-2086; EXS. 2071-2073; EX. 2007 at 125:25-126:6. The regression test files indicate the date they were run, which block was tested, which tests were used, and what the results of the tests were. Regression testing was continually conducted to ensure future versions of developed emulation and RTL code describing the critical blocks of the R400 continued to exhibit previously-developed functionality.

R400 Regress Testing:

See EXS. 2085-2086; EXS. 2071-2073.

Netlist synthesis:

228. Once feature sets were completed for the critical blocks of the R400, netlists were synthesized from stabilized versions of RTL. *E.g.*, EX. 2007 at 32:8-34:8, 101:3-12, 104:22-105:5, 114:8-12, 123:24-130:2. The netlists were generated on a block level and integrated with a chip level netlist. Diligence went into

synthesizing these blocks as multiple milestone versions of netlists were generated.

R400 IKOS Evaluation:

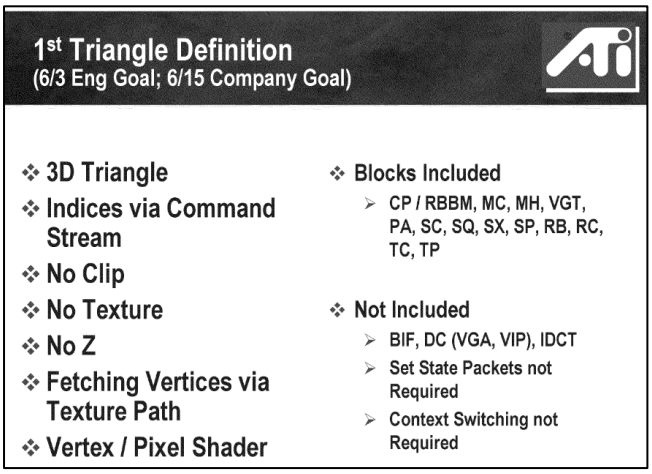
229. Once netlists were synthesized, diligence was utilized to place and route the netlist on the IKOS system for testing in a hardware environment. *E.g.*, EX. 2007 at 105:6-106:25. Once placed and routed, continual tests were run on the IKOS FPGAs embodying the synthesized functionality of the developed block level RTL. *See* EX. 2069; EX. 2087.

Triangle test passes on software emulation and Verilog simulation by at least August 30, 2002

230. A critical verification step during the R400 development was execution of the "1st Triangle Test" in both software emulation and Verilog simulation. *E.g.*, EX. 2007 at 115:22-118:7; EX. 2004 at ¶¶ 44-52. The 1st Triangle test involves the rendering of a 3D triangle. Completing this test is an indication that the features of essential blocks of the R400 operate for their intended purpose: rendering 3D graphics. Moreover, the 1st Triangle Test exercised the unified (e.g., vertex/pixel) shader. The triangle test (Gouraud shaded triangle) is executed by running functional test on hardware – this means that a low level API allowed small C++ programs to be written so as to exercise the hardware in a similar manner of expected operating conditions. *See* EX. 2102 (6/28/2017 Lefebvre Dep.), at 149:20-152:25.

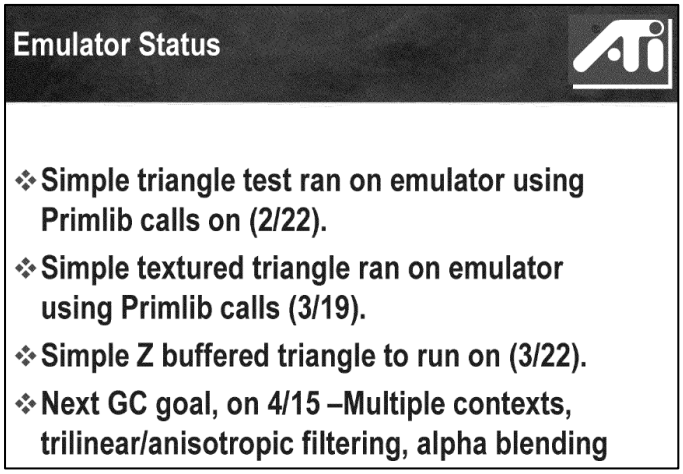
231. Passing the test also meant that the RTL was stable and would operate for its intended purpose throughout the remainder of the development lifecycle,

including IKOS prototyping. *See* EX. 2102 (6/28/2017 Lefebvre Dep.), at 114:21-117:21; 124:5-11; 125:4-23; 222:1-12, 118:9-21. Regression testing confirms that subsequent versions of RTL still would pass the triangle test. *See Id.* (6/28/2017 Lefebvre Dep.), at 126:2-16. The testing was performed for both emulation and RTL code. *See Id.* (6/28/2017 Lefebvre Dep.), at 123:10-23.



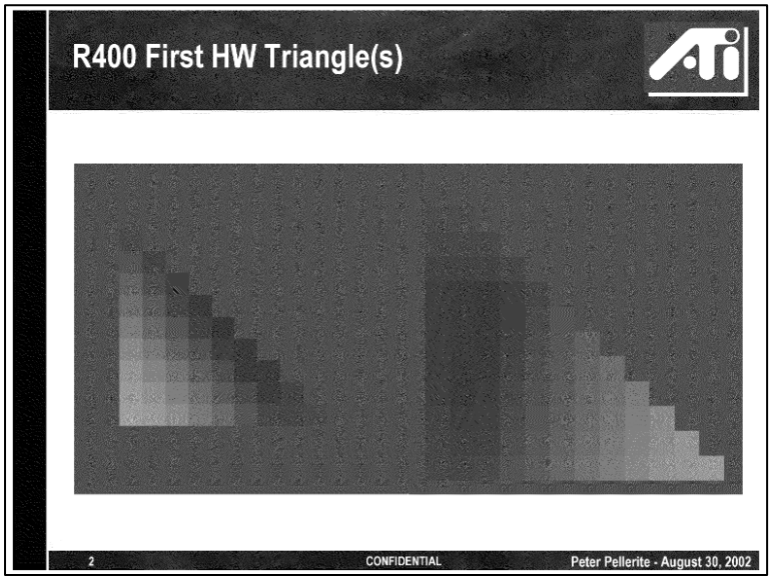
See EXS. 2110-2111; *see* EX. 2111, at p. 70 (R400 Program Review); *see also* EX. 2004 (Lefebvre Decl., IPR2015-00325), at pp. 25-32; *see generally* EX. 2118.

The 1st Triangle Test was first run in software emulation on February 22, 2002.

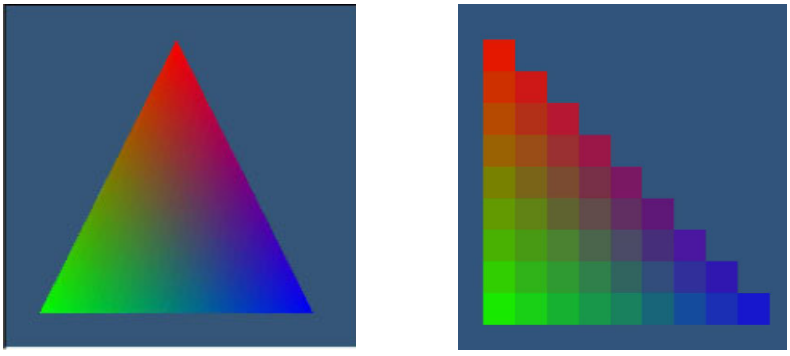


EX. 2111 (R400 Program Review), at pp. 70, 87; *see generally* EXS. 2110-2111.

On August 30, 2002, the emulator status was nearing completion as the feature sets for the essential blocks of the R400 were nearly finalized. Also on August 30, 2002, the Verilog simulations on the RTL blocks successfully passed the 1st Triangle Test, constituting what a person skilled in the relevant art would consider an actual reduction to practice.



EX 2111 (R400 Program Review), at p. 133; *see generally* EXS. 2110-2111.

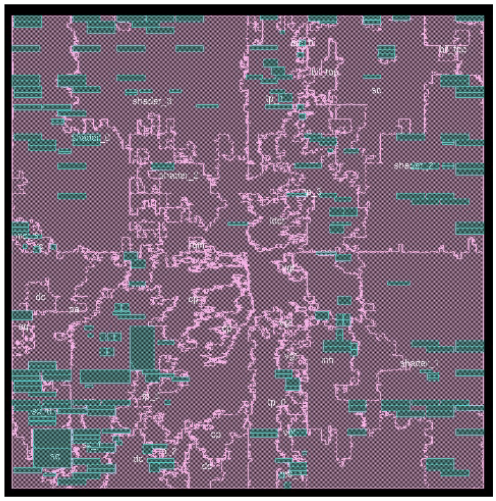


See EX. 2083, (R400 Website, Software Emulation Triangle Test, 9/6/2002); EX. 2084, (R400 Website, Verilog Simulation Triangle Test, 9/6/2002)*; see generally* EXS 2085-2086, (R400 Regress Testing).

Netlist generation

232. Netlist synthesis and generation was an important step in the development of the R400. Once generated, the netlist would be utilized by the physical design team to generate a floorplan for the chip. EX 2110 (R400 Program Reviews) at pp. 42, 45; *see generally* EXS. 2110-2111; EX. 2009. Floorplanning is the process of identifying structures that should be placed close together, and allocating space for them in such a manner as to meet the sometimes conflicting goals of available space, required performance, and the desire to have all structures close to one another. Floorplanning takes into account the macros used in the design, memory, other IP cores and their placement needs, the routing possibilities, and also the area of the entire design. Floorplanning also decides the IO structure and aspect ratio of the design. On September 16, 2002, the first floorplan was created for Netlist 2.0.

First R400 Floorplan - Silicon Perspectives
(no I/O constraints)



See EX. 2088 (R400 Program Review), at pp. 6, 9; *see generally* EXS. 2110-2111 (R400 Program Reviews).

233. Four major netlist releases were created: Netlist 2.0, Netlist 3.0, Netlist 4.0 and Netlist 4.1. *See generally* EXS. 2110-2111 (R400 Program Reviews).

Netlist placement on IKOS:

234. The IKOS apparatus is a VirtualLogic Emulation System. *See* EX. 2074 (VirtualLogic Software User's Guide), at p. 25. IKOS allows design engineers to utilize "emulation," which is a technology that creates a prototype of an application-specific integrated circuit (ASIC) design in hardware. The prototype is built through partitioning the design into smaller pieces and then mapping or compiling the design onto a large array of field programmable gate arrays (FPGAs) or custom chips.

Emulation is a technology that creates a prototype of an Application Specific Integrated Circuit (ASIC) design in hardware. The prototype is generally built through partitioning the design into smaller pieces and then mapping or compiling the design onto a large array of Field Programmable Gate Arrays (FPGAs) or custom chips. The prototype is a complete functional implementation of the design including all digital functions and memories.

The ASICs can be tested under real world operating conditions rather than using an approximation of their operating environment. In-system emulators get their stimuli directly from the target system, unlike simulations, which require test programs, testbenches, and stimulus files. The advantages of emulation include full system integration and debugging before the ASIC design is finished and the device fabricated. This is especially true of low level software, such as diagnostics and device drivers, which often require actual target systems for complete testing. The user can design and verify the ASICs, system hardware, and system software at the same time.

See EX. 2074, at p. 25; *see generally* EX. 2075 (IKOS VirtualLogic 3.5 User Guide); *see generally* EX. 2076 (IKOS VirtualLogic 2.1 Software User's Guide).

235. IKOS consists of a reconfigurable hardware system. The system is accompanied by software tools that enable the implementation and debugging of a design on the hardware. The IKOS hardware is configurable and consists of a system board, array boards, interface to logic analyzer, interface for the target system, and an internal data sampling capability.

VirtuaLogic Emulation System Components

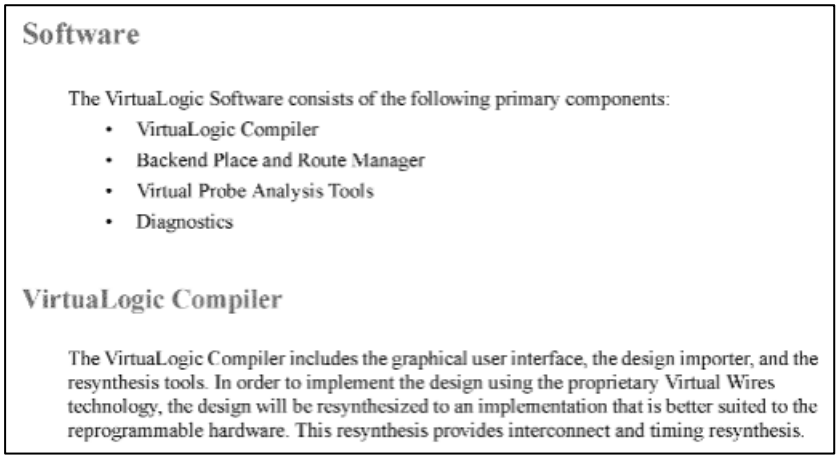
The VirtuaLogic product consists of a reconfigurable hardware system, debug hardware, and interfacing hardware. The hardware is driven by software tools used to implement and debug the ASIC design. The entire system is called the VirtuaLogic Emulation System.

Hardware

The VirtuaLogic Emulator is the primary hardware component. It is a configurable system that consists of a System Board, up to six Array Boards, an interface to a Logic Analyzer, an interface for the target system and an IDS (Internal Data Sampling) capability. When programmed with the user's design, the VirtuaLogic Emulator becomes the chip prototype.

See EX. 2074, at p. 26; *see generally* EX. 2075 (IKOS VirtualLogic 3.5 User Guide); *see generally* EX. 2076 (IKOS VirtualLogic 2.1 Software User's Guide); *see also* EX. 2100 (IKOS Schematic).

236. IKOS also includes a virtual logic compiler that imports designs and includes synthesis tools. The software receives a netlist, resynthesizes it, and provides interconnect and timing analysis for the reprogrammable hardware.



Software

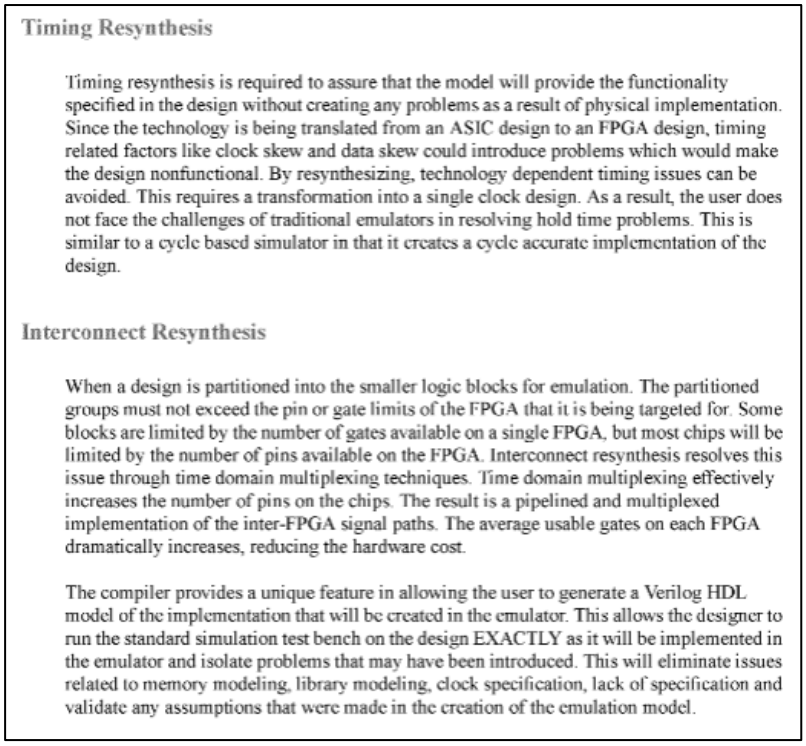
The VirtuaLogic Software consists of the following primary components:

- VirtuaLogic Compiler
- Backend Place and Route Manager
- Virtual Probe Analysis Tools
- Diagnostics

VirtuaLogic Compiler

The VirtuaLogic Compiler includes the graphical user interface, the design importer, and the resynthesis tools. In order to implement the design using the proprietary Virtual Wires technology, the design will be resynthesized to an implementation that is better suited to the reprogrammable hardware. This resynthesis provides interconnect and timing resynthesis.

See EX. 2074, at p. 28; *see generally* EX. 2075 (IKOS VirtualLogic 3.5 User Guide); *see generally* EX. 2076 (IKOS VirtualLogic 2.1 Software User's Guide).



Timing Resynthesis

Timing resynthesis is required to assure that the model will provide the functionality specified in the design without creating any problems as a result of physical implementation. Since the technology is being translated from an ASIC design to an FPGA design, timing related factors like clock skew and data skew could introduce problems which would make the design nonfunctional. By resynthesizing, technology dependent timing issues can be avoided. This requires a transformation into a single clock design. As a result, the user does not face the challenges of traditional emulators in resolving hold time problems. This is similar to a cycle based simulator in that it creates a cycle accurate implementation of the design.

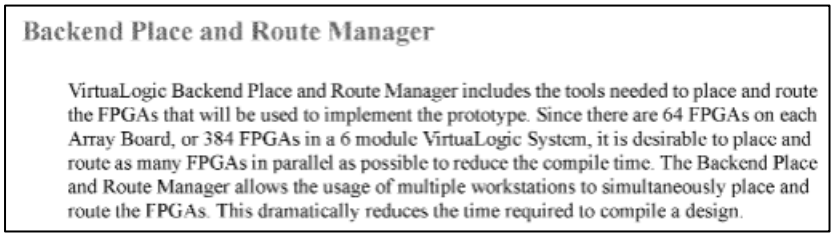
Interconnect Resynthesis

When a design is partitioned into the smaller logic blocks for emulation. The partitioned groups must not exceed the pin or gate limits of the FPGA that it is being targeted for. Some blocks are limited by the number of gates available on a single FPGA, but most chips will be limited by the number of pins available on the FPGA. Interconnect resynthesis resolves this issue through time domain multiplexing techniques. Time domain multiplexing effectively increases the number of pins on the chips. The result is a pipelined and multiplexed implementation of the inter-FPGA signal paths. The average usable gates on each FPGA dramatically increases, reducing the hardware cost.

The compiler provides a unique feature in allowing the user to generate a Verilog HDL model of the implementation that will be created in the emulator. This allows the designer to run the standard simulation test bench on the design EXACTLY as it will be implemented in the emulator and isolate problems that may have been introduced. This will eliminate issues related to memory modeling, library modeling, clock specification, lack of specification and validate any assumptions that were made in the creation of the emulation model.

See EX 2074, at p. 29; *see generally* EX. 2075 (IKOS VirtualLogic 3.5 User Guide); *see generally* EX. 2076 (IKOS VirtualLogic 2.1 Software User's Guide).

237. IKOS also includes a backend place and route manager that places and routes the FPGAs that are used to implement the prototype.



Backend Place and Route Manager

VirtuaLogic Backend Place and Route Manager includes the tools needed to place and route the FPGAs that will be used to implement the prototype. Since there are 64 FPGAs on each Array Board, or 384 FPGAs in a 6 module VirtuaLogic System, it is desirable to place and route as many FPGAs in parallel as possible to reduce the compile time. The Backend Place and Route Manager allows the usage of multiple workstations to simultaneously place and route the FPGAs. This dramatically reduces the time required to compile a design.

See EX. 2074, at p. 29; *see generally* EX. 2075 (IKOS VirtualLogic 3.5 User Guide); *see generally* EX. 2076 (IKOS VirtualLogic 2.1 Software User's Guide).

238. The following is a process flow of how netlist designs are imported into IKOS and placed and routed on the configurable FPGAs to construct a hardware

prototype.

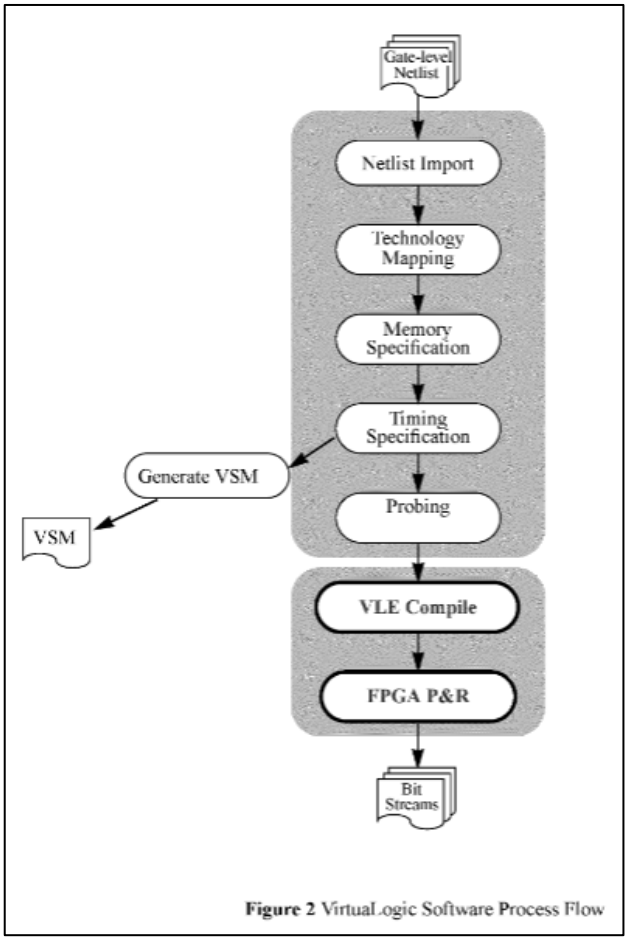


Figure 2 Virtual.ogic Software Process Flow

See EX. 2074, at p. 31; *see generally* EX. 2075 (IKOS VirtualLogic 3.5 User Guide); *see generally* EX. 2076 (IKOS VirtualLogic 2.1 Software User's Guide).

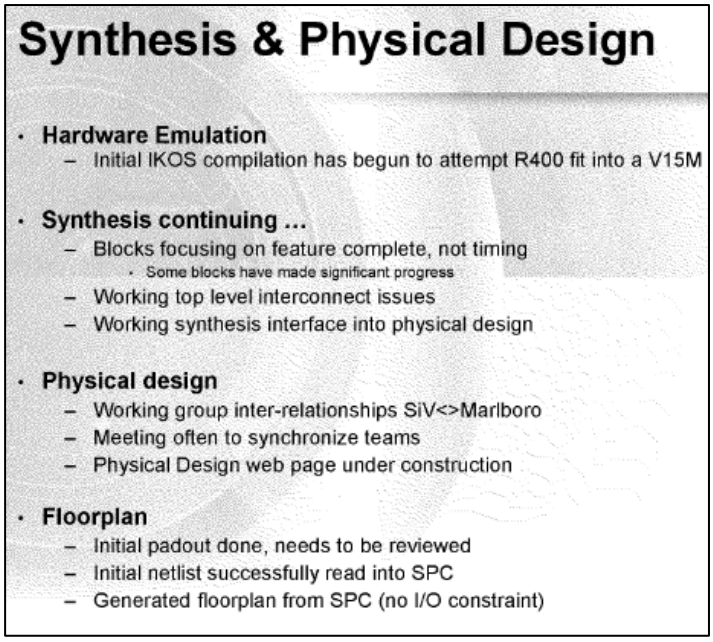
239. Information pertaining to the aspects of the IKOS prototyping environment corroborates testimony offered by named inventor, Laurent Lefebvre. In particular, Mr. Lefebvre stated that the IKOS FPGA has a netlist pushed onto it, thereby becoming a hardware chip. *See* EX. 2102 (Lefebvre Dep.), at 31:14-34:8, 49:9-53:9, 99:10-100:2. Mr. Lefebvre also confirmed that netlists are synthesized from RTL describing the R400, and that every netlist was used on the IKOS. *See*

Id. (Lefebvre Dep.), at 104:12-105:5, 109:6-9, 110:1-7. Once the netlists are placed on the IKOS environment, the IKOS emulation with software acts as a hardware emulator, and each emulation start date (as described in program reviews) indicates that an R400 netlist was successfully ported onto the IKOS environment. *See Id.* (Lefebvre Dep.), at 129:15-19, 201:19-203:2.

240. The netlists that were placed on the IKOS environment are tied to RTL, and all of this information is discernable from Perforce. *See Id.* (Lefebvre Dep.), at 104:12-105:5, 110:1-7, 111:20-112, 201:19-203:2. IKOS prototyping had been planned to start in early October 2002 and continued into 2003. *See Id.* at (Lefebvre Dep.), at 201:4-10, 214:1-23, 1005:9-106:25.

The first IKOS netlist was compiled on September 30, 2002

241. As early as September 16, 2002, RTL was synthesized and compiled for placement on the IKOS system.



See EX. 2024 (R400 Program Review, 9/16/2002), at p. 5; *see generally* EXS. 2088-2089 (R400 Program Review).

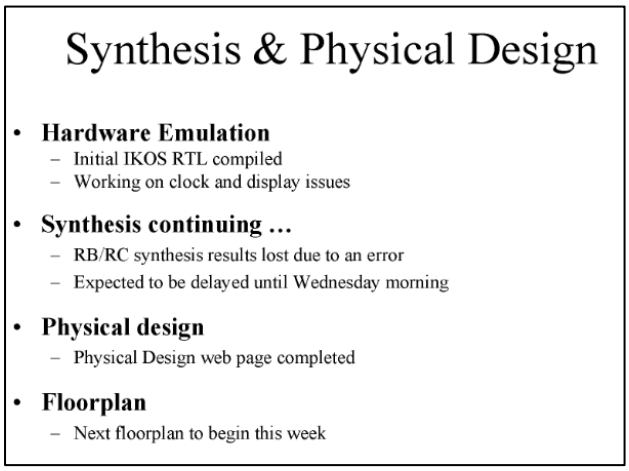
A few weeks later, on September 30, 2002, the RTL design and timing for the essential blocks of the R400 was nearly complete.

Block Logic Design and Timing

Block	Percent Feature Complete	Forecast Feature Complete	Emulator Complete	Directed Tests Written	Synthesis W/C		TNS (nS)
					SCLK (2.5nS)	MCLK (2nS) PIXCLK (2.5nS) BCLK (14.1nS)	
MC/MH*	100% / 99%	9/23 - 9/16	N/A	15%	MC:2.5; MH: 4.41	MCLK: 2.39	0 / 14491
RB*/RC	90% / 99%	9/30	95%	89%	RB: 8.2; RC 2.5		16689 / 0
SX*/SQ	95%	9/30	99%	97%	SX: 2.5; SQ: 2.51		0 / 0
SP	100%	9/16	98%		SP: 2.5		0
TP/TC*	99% / 95%	9/16	95%	69%	TP: 2.5; TC: 2.5		0 / 0
CG/CGM/ROM	100%	9/16	N/A	4%	CG: 2.5; CGM:2.5; DBG:2.5		0
PA	100%	9/16	100%	36%	PA: 2.56		143
VGT	100%	9/16	100%	70%	VGT: 2.52		0
SC*	90%	9/16	95%	68%	SC:2.52		140
RBBM	95%	9/16	N/A	90%	RBBM: 2.5		0
CP	95%	9/16	90%		CP: 2.73		408
Display - DCP	100%	9/16	100%	84%	Display: 4.8	PIXCLK: 2.66	3608
Display - LB	100%	9/16	100%	60%	3.08		
Display - SCL	100%	9/20	90%	77%	3.94		
Display - CRTC	100%	9/16	100%	59%	2.02		
Display - DISPOUT	100%	9/16	80%	62%	2.8		
Display - TVOUT	100%	9/16	30%	0%		20.77(TVCLK:15ns)	
VGA	100%	9/16	99%	65%	5.12		
VIP	100%	9/16	100%	100%	2.15		
BIF	95%	10/11	90%	17%	BIF 5.75	BCLK: 14nS; AGP:3.5nS	242
IDCT	100%	9/16	97%	100%	IDCT: 2.52		75

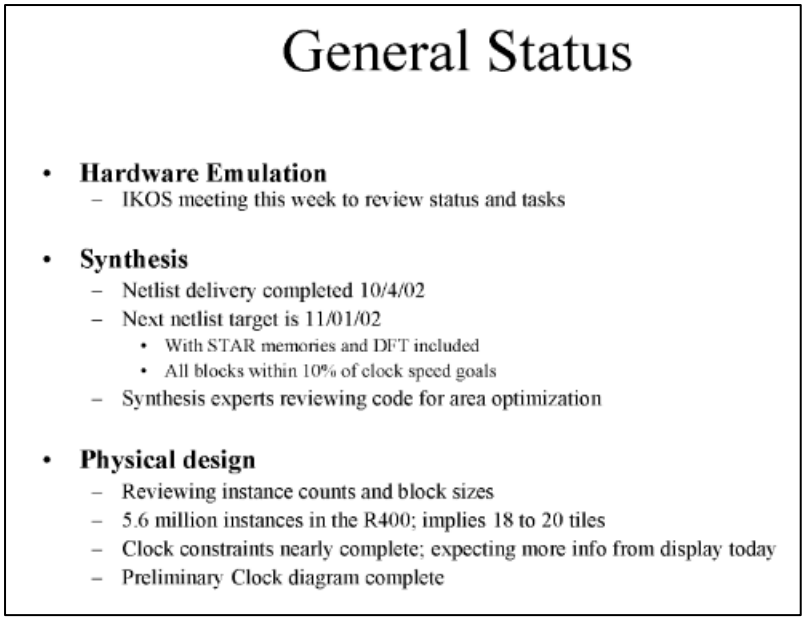
See EX. 2089 (R400 Program Review, 9/30/2002), at p. 4; *see generally* EXS. 2088-2089 (R400 Program Review).

At this point in time, the initial IKOS RTL was compiled and placed on the IKOS system.



See EX. 2089 (R400 Program Review, 9/30/2002), at p. 9*;* EX. 2041 (R400 Program Review), at p. 2; *see generally* EXS. 2088-2089 (R400 Program Review).

Subsequent iterative netlists were synthesized and placed on the IKOS system. The first milestone netlist delivery occurred on October 4, 2002. This netlist would have been synthesized from RTL existing at the same point in time.



See EX. 2089 (R400 Program Review, 10/15/2002), at pp. 30-36; *see generally* EXS. 2088-2089 (R400 Program Review).

As the following slide depicts, Netlist 1.0 was completed on August 3, 2002, Netlist 2.0 was completed on October 4, 2002, and Netlist 3.0 was completed on January 2, 2003.

R400 Program Schedule

Task	Original	Last FC	Current FC
Emulator Test template Complete	01-18-02	01-18-02	Complete
GC Emulator integration – 1 triangle	02-22-02	02-21-02	Complete
Core Emulator pixel / shader tests run	03-15-02	03-19-02	Complete
Block Testing Begins	04-16-02	05-01-02	Complete
GC/Chip Integration Start	05-17-02	05-15-02	Complete
Simulate 1 Triangle / Emulator ready for SW	06-15-02	07-01-02	Complete
First Syntheses (n-1)	07-12-02	08-03-02	Complete
Verilog Feature Complete	09-16-02	10-31-02	Complete
Second Synthesis (n)	09-23-02	10-04-02	Complete
IKOS Emulation start	10-11-02	01-13-03	Complete
Third Synthesis	11-01-02	01-02-03	Complete
Early block delivery starts	11-08-02	12-16-02	Complete
IKOS Emulation (w/ Software) begins	11-11-02	01-16-02	03-16-03
RTL Freeze / Final Netlist (Gate level ECO only)	11-15-02	01-31-03	05-01-03
A11 Base Layers Tapeout	01-10-03	03-28-03	06-26-03
A11 Metal Layers Tapeout	01-24-03	04-11-03	07-10-03
First Samples for Engineering		06-25-03	09-12-03
A12 Tapeout		07-31-03	10-17-03
A12 Samples for Engineering		09-04-03	11-28-03
R400 Customer Samples		09-11-03	12-05-03
Product Delivery		10-04-03	12-29-03

See EX. 2089 (R400 Program Review, 1/16/2003), at pp. 110-118; *see generally* EXS. 2088-2089 (R400 Program Review).

Program reviews indicate that Netlist 1.0 had, in fact, been handed off prior to August 30, 2002. This corroborates the first netlist having been synthesized on August 3, 2002.

Netlist Handoff's

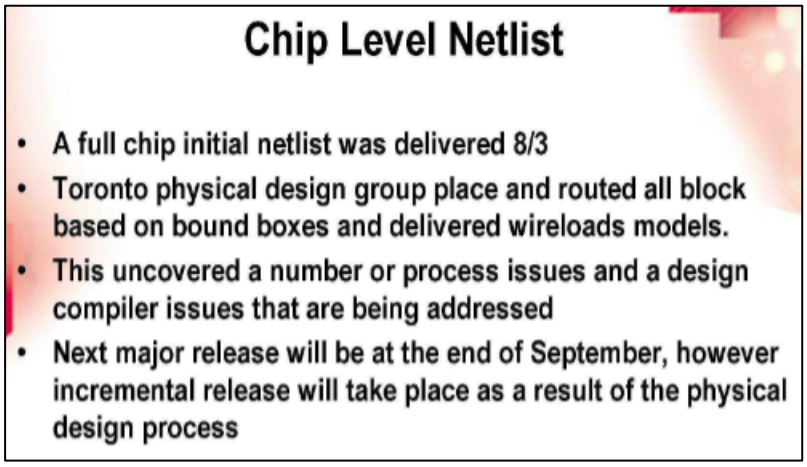
1st Netlist

- Netlist quality good enough so initial floorplan work not wasted
- Rough pad ring and analog macro placement in place.
- Clock structure in place, with some documentation
- Reasonable top level constraints
- Need libraries in-place (e.g.., preliminary lef's of all netlist components)

Deliverable from Marlboro

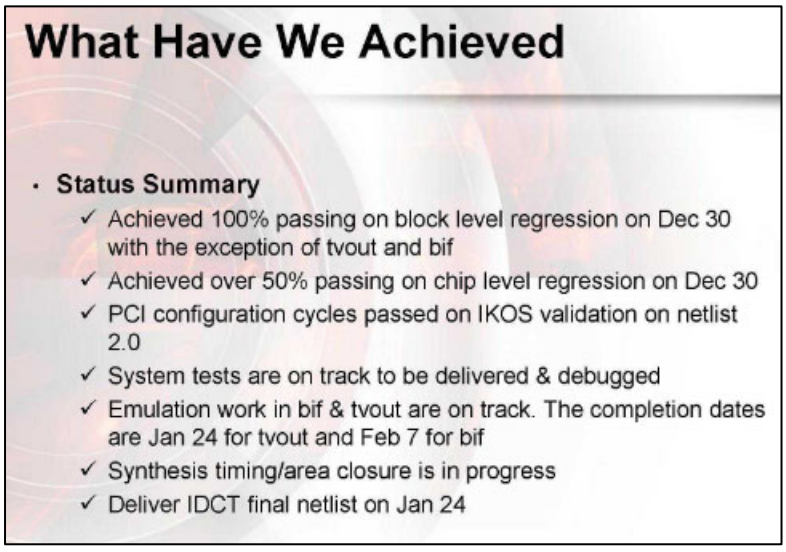
- 1st SPC pass floor-plan (Marlboro lead, SVC CAD support)

See EX. 2040 (R400 Program Review, 8/30/2002), at p. 4; *see generally* EXS. 2088-2089 (R400 Program Review).



See EX. 2041 (R400 Program Review), at p. 5.

Confirming that Netlist 2.0 was placed on the IKOS system is the fact that PCI configuration cycles passed IKOS validation.



See EX. 2033 (R400 Program Review, 1/15/2003), at p. 2.

Netlist 3.0 was synthesized and placed on IKOS no later than January 13, 2003

242. On January 2, 2003, Netlist version 3.0 was released.

R400 IKOS Status

LATEST STATUS	Diagnostics on Netlist 4.1: Many failed test cases now passed on the latest patched netlist BIOS on Netlist 4.1: Successfully boot BIOS from ROM, but some VGA modes working while some others not working - under investigation
DATE (MM/DD/YYYY)	**DETAIL STATUS**
Thu 01/02/2003	Netlist 3.0 released
Sat 01/11/2003	**[16:41 Ron White]** We have the reset block working correctly now. The target PC boots and the R400 chip is responding to configuration reads and writes. Operations to non-pci configuration registers fail. We are investigating the problem.
Wed 01/15/2003	**[15:23 Ron White]** Thanks to Gabe, we found another mistake in the Star memory models we created for the R400. The models are fixed and we are recompiling. We'll test it after supper. **[22:42 Ron White]** At least some of the R400 registers are working in the latest IKOS compile. (BIOS scratch and VGA) We will test the registers more extensively tomorrow morning. The next step is to get the memory interface to work. Is it worth having a BIOS guy start his work on a second R400 setup if the memory is not working yet ?

See EX. 2087 (R400 IKOS Status), at p. 1.

Subsequent to its release, it was placed on the IKOS system, and emulation testing began on January 13, 2003.

R400 Program Schedule

Task	Original	Last FC	Current FC
Emulator Test template Complete	01-18-02	01-18-02	Complete
GC Emulator integration – 1 triangle	02-22-02	02-21-02	Complete
Core Emulator pixel / shader tests run	03-15-02	03-19-02	Complete
Block Testing Begins	04-16-02	05-01-02	Complete
GC/Chip Integration Start	05-17-02	05-15-02	Complete
Simulate 1 Triangle / Emulator ready for SW	06-15-02	07-01-02	Complete
First Syntheses (n-1)	07-12-02	08-03-02	Complete
Verilog Feature Complete	09-16-02	10-31-02	Complete
Second Synthesis (n)	09-23-02	10-04-02	Complete
IKOS Emulation start	10-11-02	01-13-03	Complete
Third Synthesis	11-01-02	01-02-03	Complete
Early block delivery starts	11-08-02	12-16-02	Complete
IKOS Emulation (w/ Software) begins	11-11-02	01-16-02	03-16-03
RTL Freeze / Final Netlist (Gate level ECO only)	11-15-02	01-31-03	05-01-03
A11 Base Layers Tapeout	01-10-03	03-28-03	06-26-03
A11 Metal Layers Tapeout	01-24-03	04-11-03	07-10-03
First Samples for Engineering		06-25-03	09-12-03
A12 Tapeout		07-31-03	10-17-03
A12 Samples for Engineering		09-04-03	11-28-03
R400 Customer Samples		09-11-03	12-05-03
Product Delivery		10-04-03	12-29-03

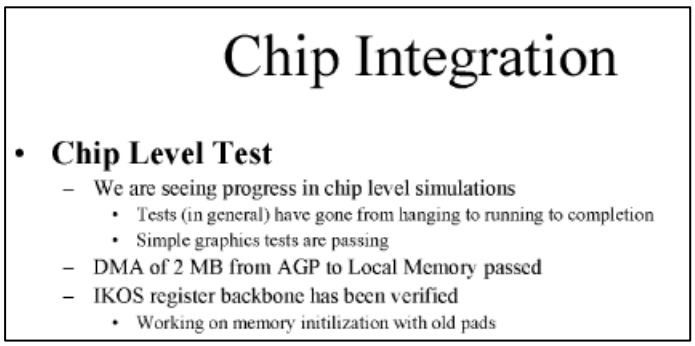
See EX 2089 (R400 Program Review, 1/16/2003) at pp. 110-118; *see generally* EXS. 2088-2089 (R400 Program Review).

On January 16, 2003, it had been established that the best way of getting an operational memory controller and memory pad for IKOS was to use RTL that had been subjected to regression testing, compile it, and place it on IKOS.

Thu 01/16/2003	**[10:48 Gabriel Abarca]** Frank Hering was just suggesting Ron that the best way of getting an operational MC and memory pads for IKOS was to get the RTL we use for regression and compile in to IKOS. Ron was saying he would like to try the idea, as this would give him both things in one shot. Please correct me if I am wrong, but I believe the most stable path to get the good MC and pad RTL models is: /proj/crayola_vol3_nobackup/run_regress/crayola_sim_chip_247_linux_rtl/devel

See EX. 2087 (R400 IKOS Status), at p. 1.

On January 27, 2003, it was confirmed that the IKOS register backbone was verified. The R400 netlist placed on the IKOS system, accordingly, could operate with the functioning registers.



See EX. 2089 (R400 Program Review, 1/27/2003) at pp. 135-149; *see generally* EXS. 2088-2089 (R400 Program Review).

Moreover, at this point in time, the feature set for the critical RTL blocks of the R400 had been completed.

Block Logic Design and Timing

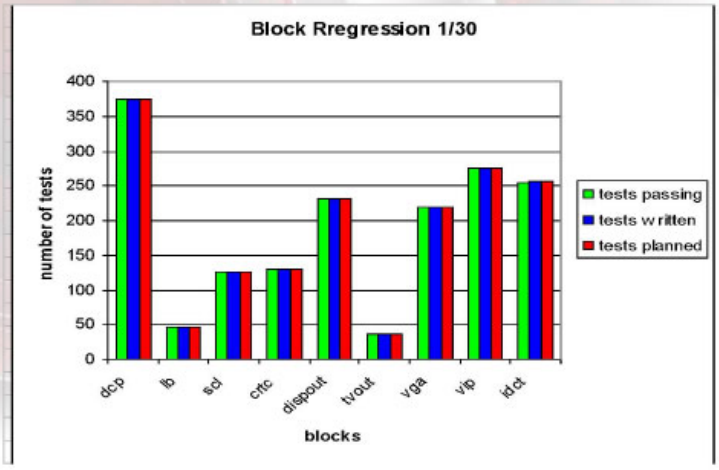
Block	Percent Feature Complete	Forecast Feature Complete	Emulator Complete	Directed Tests Written	Synthesis W/C		TNS (nS)
					SCLK (2.25nS)	MCLK (1.67nS), PIXCLK (2.5nS), BCLK (14nS), AGP(3.5nS), TVCLK(10nS)	
MC/MH	100%	Done	N/A	100%	MC:2.84; MH: 2.40	MC - MCLK: 1.71	44 / 93
AB/RB/RC	100%	Done	100%	80%	AB: 2.46; RB: 3.79; RC 2.25		0.3 / 7983 / 0
SX/SQ	100%	Done	100%	96%	SX: 2.57; SQ: 3.09		36 / 136
SP	100%	Done	100%	100%	SP: 2.70		1560
TP/TCM/TCF/TCR	100%	Done	100%	86%	TP: 3.13; TCM: 2.25; TCF: 4.24; TCR:3.32		523 / 0 / 18168 / 2225
CG/CGM/ROM	100%	Done	N/A	96%	CG: 2.25; CGM:2.25; DBG 2.25		0
PA	100%	Done	100%	97%	PA: 2.27		2
VGT	100%	Done	100%	98%	VGT: 2.29		2
SC / SC_B	100%	Done	100%	87%	SC: 2.34 / SC_B: 2.26		35 / 0.1
RBBM	100%	Done	100%	100%	RBBM: 2.28		0.4
CP	100%	Done	100%		CP: 2.25		0
Display - DCP	100%	Done	100%	100%	3.58		212
Display - LB	100%	Done	100%	100%			
Display - SCL	100%	Done	100%	100%	2.52		
Display - CRTC	100%	Done	100%	100%	2.6	PIXCLK: 2.66	
Display - DISPOUT	100%	Done	100%	100%	3.86		
Display - TVOUT	100%	Done	100%	68%	2.29		
VGA	100%	Done	100%	100%	3.59		
VIP	100%	Done	100%	100%	2.92		
BIF	96%	2/15	90%	64%	BIF 2.92	BCLK: 14nS; AGP:3.50nS	142
IDCT	100%	Done	100%	100%	IDCT: 2.30		2

See EX. 2089 (R400 Program Review, 1/27/2003) at pp. 135-149; *see generally* EXS. 2088-2089 (R400 Program Review).

By February 12, 2003, RTL regression was completed.

What Have We Achieved (block RTL regression)

- All blocks are 100% passing



See EX. 2032 (R400 Program Review, 2/12/2003), at p, 5; *see generally* EXS. 2088-2089 (R400 Program Review).

By February 10, 2003, the Netlist 3.0 was embodied in an IKOS system that had functioning register reads and writes, memory writes and reads, and basic display.

IKOS progressing
- Register R/W working
- Memory writes working, memory reads working on one channel
- Basic Display test are ready for IKOS

See EX. 2089 (R400 Program Review, 2/10/2003,) at pp. 150-162; *see generally* EXS. 2088-2089 (R400 Program Review).

On March 6, 2003, test results of Netlist 3.0 were producing expected results and the program looked on track. This result indicates that the critical feature set of the RTL was operative in the IKOS hardware system.

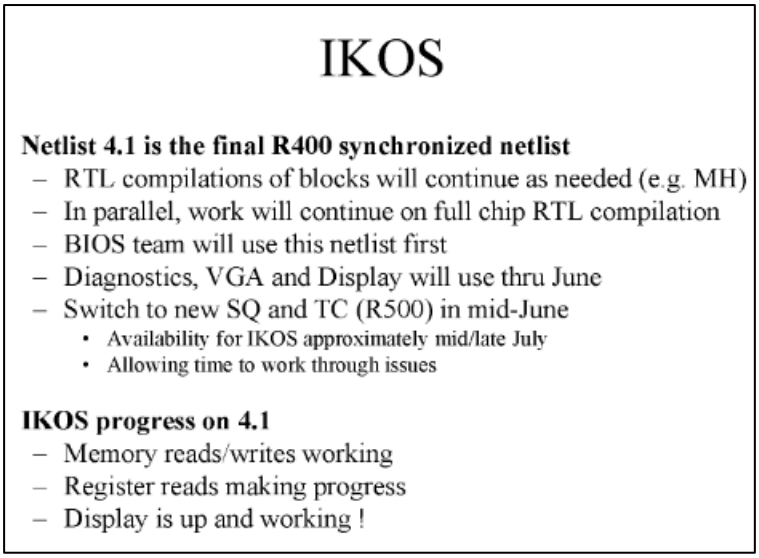
Thu 03/06/2003	**[11:04 Brian Leblanc]** I have logged off the Ikos box. I was debugging the register test using indexed accesses and writing and reading using 8/16/32 bit accesses. The tests produced the expected results for the 3.0 net list and the program is starting to look pretty good.

See EX. 2087 (R400 IKOS Status), at p. 5.

Netlist 4.1 was synthesized and placed on IKOS as of March 24, 2003

243. Netlist 4.1 was the final R400 synchronized netlist. As of March 24, 2003, Netlist 4.1 was synthesized and placed in the IKOS hardware system. As previously stated, RTL regression had been completed at this point. *See* EX. 2032 (R400 Program Review, 2/12/2003), at p. 4. Further, memory reads and writes were functioning and the display of the IKOS system operated properly.

ATI Ex. No 2121
IPR2023-00922
Page 160 of 324



IKOS

Netlist 4.1 is the final R400 synchronized netlist

- RTL compilations of blocks will continue as needed (e.g. MH)
- In parallel, work will continue on full chip RTL compilation
- BIOS team will use this netlist first
- Diagnostics, VGA and Display will use thru June
- Switch to new SQ and TC (R500) in mid-June
  - Availability for IKOS approximately mid/late July
  - Allowing time to work through issues

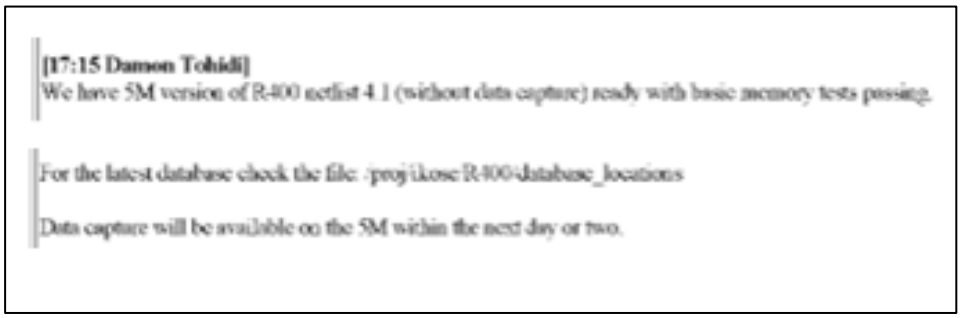
IKOS progress on 4.1

- Memory reads/writes working
- Register reads making progress
- Display is up and working !

See EX. 2089 (R400 Program Review, 3/24/2003) at pp. 181, 188, 190; *see generally* EXS. 2088-2089 (R400 Program Review).

Wed 03/19/2003	**[13:32 Colin Steward]** We have Netlist 4.1 ready with basic memory tests passing. For the latest database check the file: /proj/ikose/R400/database_locations As always make sure you schedule time using the calendar. We are working on having a 5M database ready shortly. Let us know if there are any problems.

See EX. 2087 (R400 IKOS Status), at p. 9.



[17:15 Damon Tohidi]
We have 5M version of R400 netlist 4.1 (without data capture) ready with basic memory tests passing.

For the latest database check the file: /proj/ikose/R400/database_locations

Data capture will be available on the 5M within the next day or two.

Fri 03/21/2003	**[16:07 Paul d'Entremont]** No issues. Ran thru test_WriteB with mem size of 0xffc00 stopped uncompleted test after 4 hours memory test run forever on ikos. Plan to run longer tests over weekend Ran with 18Mar_r400_net4.1_30M_lili_stub_fix_..._new_NMDQSS_pad.vms

See d. at pp. 11-12

Tue 03/25/2003	**[09:31 Paul d'Entremont]** Results for Craymem testing overnight testing. Used Netlist 4.1 with beifix10.db & with R400 IKOS speed of 200kHz. Notes: 1) Applied workaround in openapp.c & pcigart.c, temporarily commented out PGL_chip_space_rgns_disjoint() function calls 2) Modified dtresult.c, TEST_highlightstring to prevent Failed status in blinking RED. 3) database /net/sunfs8/sunfs8_nobackup6/local_user/ikos/r400/ 19Mar_r400_net4.1_30M_lili_stub_fix_..._data_capt.vmw Increased IKOS speed appears to have helped memory testing, refresh rates etc. Basic memory tests status: Fill PASS Data Line Toggle PASS Data Mask Toggle PASS Data & Address Line Toggle PASS Read Write Column PASS

[11:38 Damon Tohidi]
The R400 IKOS Database with the new mh and MC's NOT disabled passes our basic tests.
Here's the location,

/net/sunfs8/sunfs8_nobackup6/local_user/ikose/r400/
16Apr_r400_net4.1_30M_..._9Apr_new_mh_rtl_apr15_MC_ROM_dis_tied_low.vmw

[12:32 Paul d'Entremont]
Memory testing with the above database allows memory access over the full range of 128M.

I will continue with more in depth memory testing when I gain access to IKOS.

[13:09 Gina Seto]
Successful capture of 640x480 display 8bpp indexed mode: IKOS_ScreenCapture_BasicDisplay(StdPalette8bpp).bmp

Checksum matched reference checksum.

database: /net/sunfs8/sunfs8_nobackup6/local_user/ikose/r400/
16Apr_r400_net4.1_30M_..._9Apr_new_mh_rtl_apr15_MC_ROM_dis_tied_low.vmw
script: beifix10.db

[16:14 Gina Seto]
Setup:
database: /net/sunfs8/sunfs8_nobackup6/local_user/ikose/r400/
16Apr_r400_net4.1_30M_lili_stub_basedon_9Apr_new_mh_rtl_apr15_MC_ROM_dis_tied_low.vmw
script: beifix10.db

Summary:
All tests passed (checksums match reference) for a very basic display using 8bpp, 16bpp, 32bpp (and some 64bpp) on the standard 256 entry palette or the PWL palette.

See Id. at p. 12.

[11:38 Damon Tohidi]
The R400 IKOS Database with the new mh and MC's NOT disabled passes our basic tests.
Here's the location.

/net/sunfs8/sunfs8_nobackup6/local_user/ikose/r400/
16Apr_r400_net4.1_30M_..._9Apr_new_mh_rtl_apr15_MC_ROM_dis_tied_low.vmw

[12:32 Paul d'Entremont]
Memory testing with the above database allows memory access over the full range of 128M.

I will continue with more in depth memory testing when I gain access to IKOS.

[13:09 Gina Seto]
Successful capture of 640x480 display 8bpp indexed mode: IKOS_ScreenCapture_BasicDisplay(StdPalette8bpp).bmp

Checksum matched reference checksum.

database: /net/sunfs8/sunfs8_nobackup6/local_user/ikose/r400/
16Apr_r400_net4.1_30M_..._9Apr_new_mh_rtl_apr15_MC_ROM_dis_tied_low.vmw
script: beifix10.db

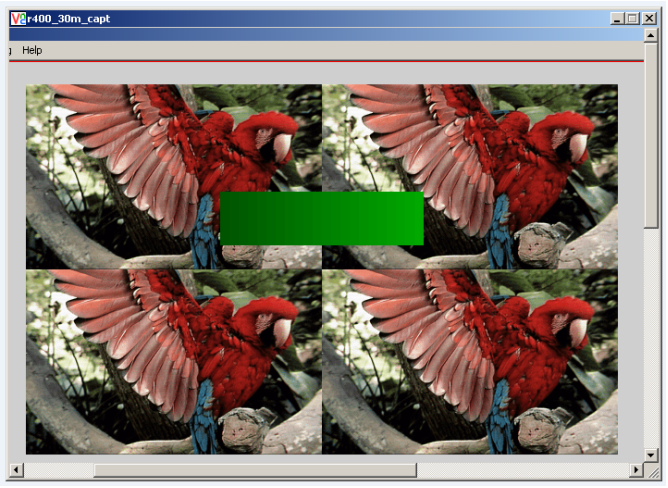
[16:14 Gina Seto]
Setup:
database: /net/sunfs8/sunfs8_nobackup6/local_user/ikose/r400/
16Apr_r400_net4.1_30M_lili_stub_basedon_9Apr_new_mh_rtl_apr15_MC_ROM_dis_tied_low.vmw
script: beifix10.db

Summary:
All tests passed (checksums match reference) for a very basic display using 8bpp, 16bpp, 32bpp (and some 64bpp) on the standard 256 entry palette or the PWL palette.

See Id. at p. 20.

IKOS screen captures

244. Additionally, once netlists were placed on the IKOS system, testing was conducted. The testing included display tests. The following is a graphic rendered and screen captured on the IKOS display.



See EXS. 2077-2078 (IKOS Screen Captures); *see also*, EX. 2087 (R400 IKOS Status).

X. CONSTRUCTIVE REDUCTION TO PRACTICE

245. The claimed inventions were constructively reduced to practice by the filing of patent applications on November 20, 2003. The 13/109,738 application was also filed on November 20, 2003 and later was granted as US Patent No. 8,760,454. The patent claims priority to US Patent No. 6,897,871 (the "'871 Patent") by way of a chain of continuation applications. US Patent No. 6,897,871 was filed as application number 10/718,318. The 10/718,318 application is identical to the 14/614,967 and 13/109,738 applications. Accordingly, the 10/718,318 application is a priority document for purposes of establishing a constructive reduction to practice of the 8,760,454 Patent.

246. The priority document (*e.g.*, US Patent Application 10/718,318) maps to each claimed element of the '454 Patent. As US Patent Application 10/718,318 resulted in the '871 Patent, the '871 Patent specification and drawings will be cited for purposes of establishing that the priority documents provides a sufficient written description for the '454 Patent. The '454 Patent specification is identical to the '871 Patent specification.

> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**. The instruction store **98** then passes the corresponding control signals to the processor **96** on line **01** to perform such vertex operations. However, if the general purpose register block **92** does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter **64** is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor **96** are continued, based on the instructions maintained in the instruction store **98**, until enough registers within the general purpose register block **92** become available. Thus, through the sharing of resources within the unified shader **62**, processing of image data is enhanced as there is no down time associated with the processor **96**.

See EX. 2080 ('871 Patent) at 5:14-34. Additionally, the '454 Patent has similar claims to the '871 Patent. For example, Claim 1 of the '871 Patent claims:

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

Id., Claim 1. This is similar to the claims of the '454 Patent, which all claim a unified shader comprising a processor operative to perform vertex calculation operations and pixel calculation operations.

A. ’454 Patent

1. Claim 1

a. “A method carried out by a unified shader comprising:”

247. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of carrying out a method to perform vertex and pixel operation in an efficient manner.

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

See EX. 2080 (’871 Patent) at 2:36-49.

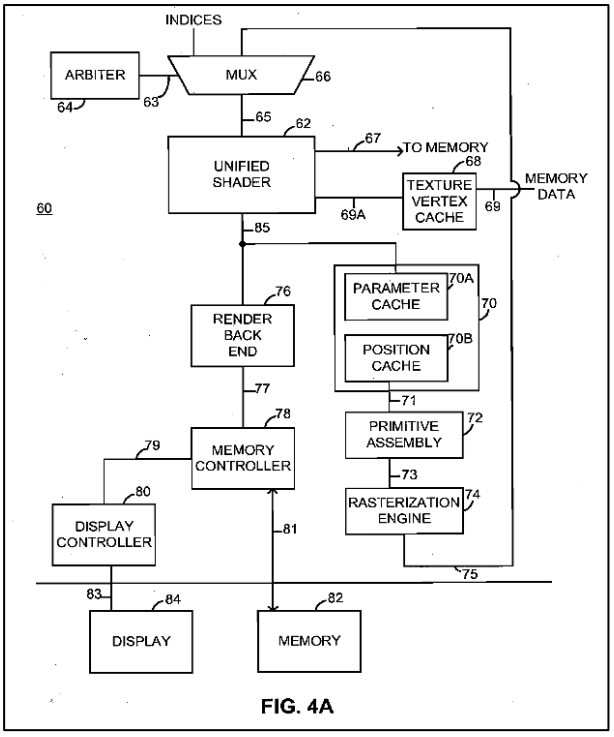


FIG. 4A

See EX. 2080 ('871 Patent) at Figure 4A.

b. "performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and"

248. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of carrying out a method to perform vertex and pixel operations.

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

See EX. 2080 ('871 Patent) at 2:36-49.

249. The unified shader includes a general purpose register block for storing the information transmitted from the multiplexer 66 or any other information to be maintained within the unified shader.

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

See EX. 2080 ('871 Patent) at 4:9-20.

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

See EX. 2080 ('871 Patent) at 5:31-51.

c. "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"

250. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations. If the general purpose register block does not have enough available space to store the incoming vertex data, such information will not be transmitted, and any pixel calculation operations are continued until enough registers within the general purpose register block become available.

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

See EX. 2080 ('871 Patent) at 5:31-51.

d. "continuing pixel calculation operations that are to be or are currently being performed by the processor"

251. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations. If the general purpose register block does not have enough available space to store the incoming vertex data, such information will not be transmitted, and any pixel calculation operations are continued by the unified shader until enough registers within the general purpose register block become available.

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

See EX. 2080 ('871 Patent) at 5:31-51.

e. "based on instructions maintained in an instruction store until enough registers within the general purpose register block become available."

252. The 13/109,738 application provides a written description for this element. For example, if the general purpose register block does not have enough available space to store the incoming vertex data, such information will not be transmitted, and any pixel calculation operations are continued by the unified shader based on the instructions maintained in the instruction store until enough registers within the general purpose register block become available.

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

See EX. 2080 ('871 Patent) at 5:31-51.

2. Claim 2

a. "A unified shader"

253. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of performing vertex and pixel operation in an efficient manner.

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

See EX. 2080 ('871 Patent) at 2:36-49.

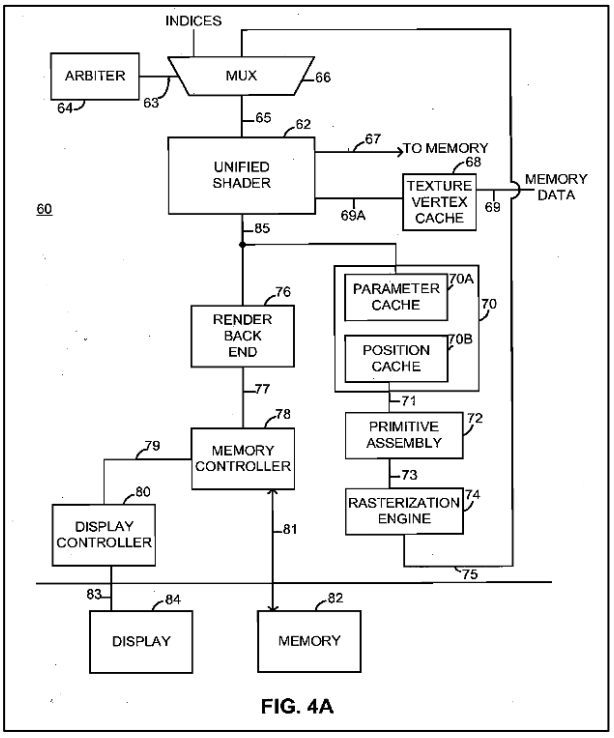


FIG. 4A

See EX. 2080 ('871 Patent) at Figure 4A.

b. "a general purpose register block for maintaining data"

254. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a

general purpose register block and a plurality of source registers that are operatively coupled to the processor of the unified shader.

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

See EX. 2080 ('871 Patent) at 4:9-20.

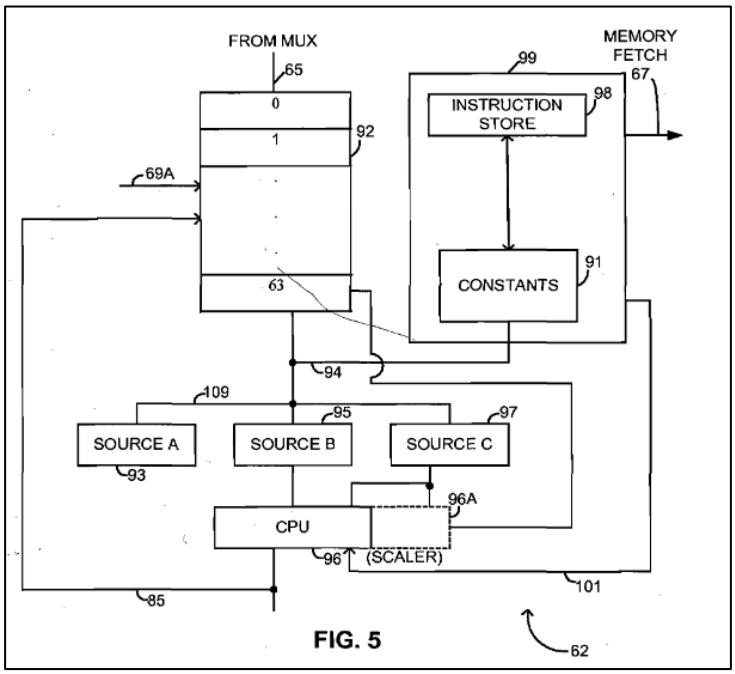


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

c. "a processor unit"

255. The 13/109,738 application provides a written description for this

element. The application describes the unified shader as including a processor.

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

See EX. 2080 ('871 Patent) at 4:9-20.

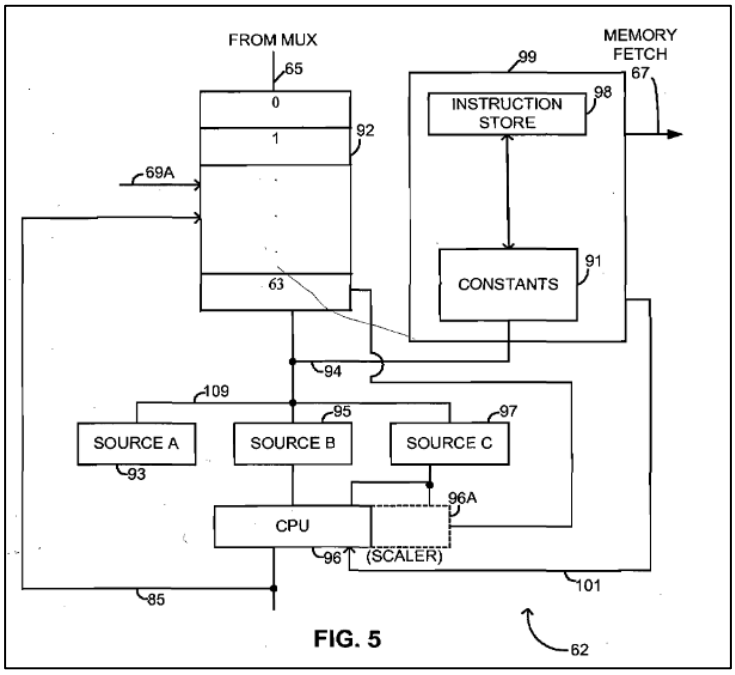


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

The processor can perform floating point arithmetic operations as well as logical operations on corresponding operands. The processor is logically partitioned into

two sections including a portion able to perform floating point operations and a portion capable of performing scalar operations.

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

See EX. 2080 ('871 Patent) at 4:21-32.

Further, the processor of the unified shader can perform both vertex and pixel operations. Specifically, the processor of the unified shader has the ability to simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

> As discussed above, the instruction store **98** maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader **99** of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader **62** of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.
>
> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 ('871 Patent) at 5:1-20.

d. "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"

256. The 13/109,738 application provides a written description for this element. For example, the application provides that the shader includes a general purpose register block for storing the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations, and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs according to the instruction maintained in the sequencer.

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

See EX. 2080 ('871 Patent) at 2:50-63.

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

See EX. 2080 ('871 Patent) at 4:9-20.

The sequencer, general purpose register, and processor unit are coupled.

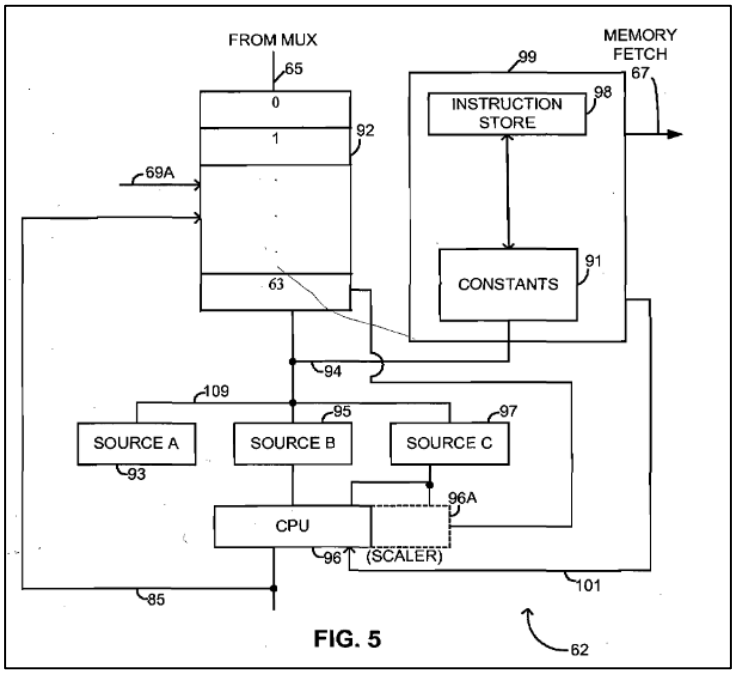


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

257. The sequencer maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block. For example the sequencer

includes an instruction store which maintains the instructions necessary for the processor to execute arithmetic and logic operations on the data maintained in the general purpose register block. As previously described, the processor performs pixel and vertex operations.

> The sequencer **99** includes constants block **91** and an instruction store **98**. The constants block **91** contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store **98** contains the necessary instructions that are executed by the processor **96** in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block **92** as provided by the source registers **93–95**. The instruction store **98** further includes memory fetch instructions that, when executed, causes the unified shader **62** to fetch texture and other types of data, from memory **82** (FIG. **4**A). In operation, the sequencer **99** determines whether the next instruction to be executed (from the instruction store **98**) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer **99** sends the request to a fetch block (not shown) which retrieves the required information from memory **82** (FIG. **4**A). The retrieved information is then transmitted to the sequencer **99**, through the vertex texture cache **68** (FIG. **4**A) as described in greater detail below.
>
> If the next instruction to be executed is an arithmetic or logical instruction, the sequencer **99** causes the appropriate operands to be transferred from the general purpose register block **92** into the appropriate source registers (**93**, **95**, **97**) for execution, and an appropriate signal is sent to the processor **96** on line **101** indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor **96** executes the instructions on the operands present in the source registers and provides the result on line **85**. The information present on line **85** may be transmitted back to the general purpose register block **92** for storage, or transmitted to succeeding components of the graphics processor **60**.

See EX. 2080 (‘871 Patent) at 4:33-67, 2:36-63.

e. "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."

258. The 13/109,738 application provides a written description for this element. For example, the processor of the unified shader can perform both vertex and pixel operations. Specifically, the processor of the unified shader has the ability to simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

> As discussed above, the instruction store **98** maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader **99** of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader **62** of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.
>
> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 ('871 Patent) at 5:1-20.

259. As previously described, the sequencer includes an instruction store which maintains the instructions necessary for the processor to execute arithmetic and logic operations on the data maintained in the general purpose register block.

As previously described, the processor performs pixel and vertex operations.

> The sequencer **99** includes constants block **91** and an instruction store **98**. The constants block **91** contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store **98** contains the necessary instructions that are executed by the processor **96** in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block **92** as provided by the source registers **93–95**. The instruction store **98** further includes memory fetch instructions that, when executed, causes the unified shader **62** to fetch texture and other types of data, from memory **82** (FIG. **4**A). In operation, the sequencer **99** determines whether the next instruction to be executed (from the instruction store **98**) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer **99** sends the request to a fetch block (not shown) which retrieves the required information from memory **82** (FIG. **4**A). The retrieved information is then transmitted to the sequencer **99**, through the vertex texture cache **68** (FIG. **4**A) as described in greater detail below.
>
> If the next instruction to be executed is an arithmetic or logical instruction, the sequencer **99** causes the appropriate operands to be transferred from the general purpose register block **92** into the appropriate source registers (**93**, **95**, **97**) for execution, and an appropriate signal is sent to the processor **96** on line **101** indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor **96** executes the instructions on the operands present in the source registers and provides the result on line **85**. The information present on line **85** may be transmitted back to the general purpose register block **92** for storage, or transmitted to succeeding components of the graphics processor **60**.

See EX. 2080 (‘871 Patent) at 4:33-67, 2:36-63.

260. When the unified shader performs vertex shading, it applies a transformation position matrix to the input position information and generates data representing a prospectively corrected image of the object to be rendered.

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

See EX. 2080 ('871 Patent) at 3:1-18.

When the unified shader performs pixel operations, it will generate color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels.

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

See EX. 2080 ('871 Patent) at 3:33-51.

3. Claim 3

a. "A unified shader"

261. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of performing vertex and pixel operation in an efficient manner.

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

See EX. 2080 ('871 Patent) at 2:36-49.

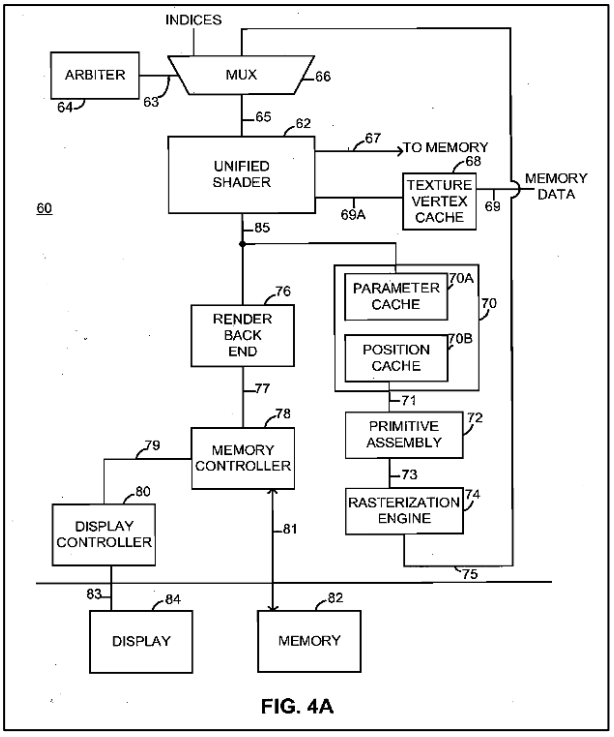


FIG. 4A

See EX. 2080 ('871 Patent) at Figure 4A.

b. "a processor unit operative to perform vertex calculation operations and pixel calculation operations"

262. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a processor.

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

See EX. 2080 (‘871 Patent) at 4:9-20.

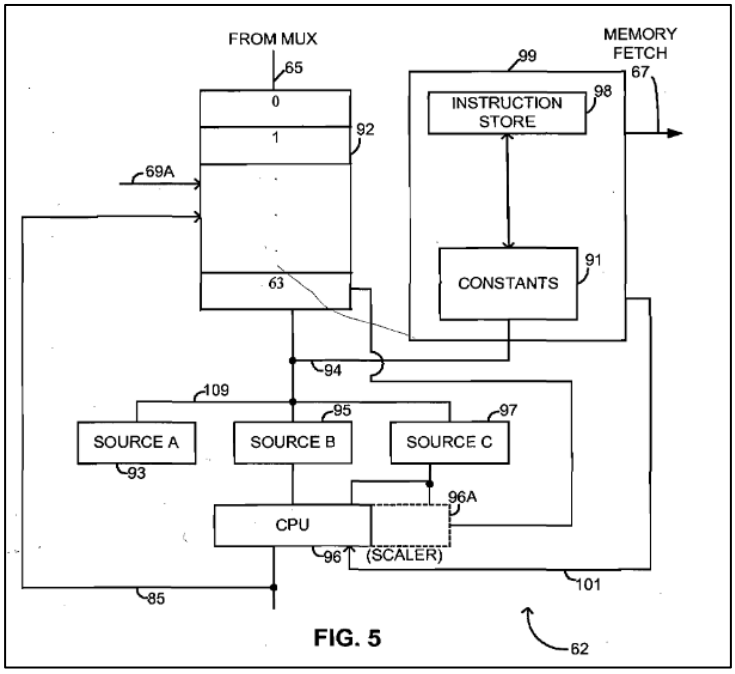


FIG. 5

See EX. 2080 (‘871 Patent) at Figure 5.

The processor can perform floating point arithmetic operations as well as logical operations on corresponding operands. The processor is logically partitioned into two sections including a portion able to perform floating point operations and a portion capable of performing scalar operations.

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

See EX. 2080 (‘871 Patent) at 4:21-32.

Further, the processor of the unified shader can perform both vertex and pixel operations. Specifically, the processor of the unified shader has the ability to simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

> As discussed above, the instruction store **98** maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader **99** of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader **62** of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.
>
> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 (‘871 Patent) at 5:1-20.

c. “shared resources, operatively coupled to the processor unit;”

263. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a general purpose register block and a plurality of source registers that are operatively coupled to the processor of the unified shader.

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

See EX. 2080 ('871 Patent) at 4:9-20.

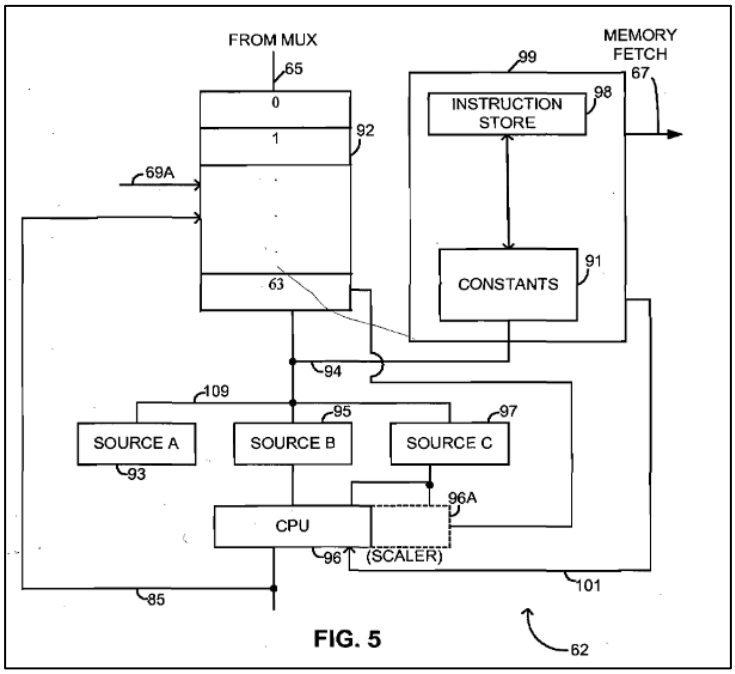


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

264. The general purpose register block can store vertex or pixel data as it receives pixel or vertex data from the multiplexer.

a sequencer **99**. The general purpose register block **92** includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer **66** on line **65** or any other information to be maintained within the unified shader. The data present in the general purpose register block **92** is transmitted to the plurality of source registers via line **109**.

See EX. 2080 ('871 Patent) at 4:14-20.

265. The multiplexer, itself, receives vertex data or pixel data.

Referring now to FIG. **4A**, in an exemplary embodiment, the graphics processor **60** of the present invention includes a multiplexer **66** having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine **74** provided at a second input. A control signal generated by

See EX. 2080 ('871 Patent) at 3:60-65.

If the general purpose register block does not have enough available space to store incoming vertex data, the vertex data will not be transmitted and pixel calculation operations will either be continued or started. Through the sharing of resources within the unified shader, processing of image data is enhanced, as there is no down time associated with the processor.

> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**. The instruction store **98** then passes the corresponding control signals to the processor **96** on line **01** to perform such vertex operations. However, if the general purpose register block **92** does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter **64** is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor **96** are continued, based on the instructions maintained in the instruction store **98**, until enough registers within the general purpose register block **92** become available. Thus, through the sharing of resources within the unified shader **62**, processing of image data is enhanced as there is no down time associated with the processor **96**.

See EX. 2080 ('871 Patent) at 5:14-34.

d. "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."

266. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a processor that can perform floating point arithmetic operations as well as logical operations on corresponding operands.

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

See EX. 2080 ('871 Patent) at 4:21-32.

Further, the processor unit uses operands from the general purpose register block (into the source registers) to execute instructions. The result of executing the operation may be transmitted back to the general purpose register block.

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

See EX. 2080 ('871 Patent) at 4:21-32.

As described previously, the processor of the unified shader can perform both vertex and pixel operations. Specifically, the processor of the unified shader has the ability to simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

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

The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 ('871 Patent) at 5:1-20.

Moreover, the processor in the unified shader will perform pixel calculation operations until enough registers within the general purpose register block become available. Then the unified shader will perform vertex calculation operations.

> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**. The instruction store **98** then passes the corresponding control signals to the processor **96** on line **01** to perform such vertex operations. However, if the general purpose register block **92** does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter **64** is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor **96** are continued, based on the instructions maintained in the instruction store **98**, until enough registers within the general purpose register block **92** become available. Thus, through the sharing of resources within the unified shader **62**, processing of image data is enhanced as there is no down time associated with the processor **96**.

See EX. 2080 ('871 Patent) at 5:14-34.

267. For example the arbiter employs an arbitration scheme as follows: the vertex data on the first input of the multiplexer is transmitted to the unified shader if there are enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader for further processing.

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

See EX. 2080 ('871 Patent) at 3:60-4:8.

4. Claim 4

a. "A unified shader"

268. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of performing vertex and pixel operation in an efficient manner.

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

See EX. 2080 ('871 Patent) at 2:36-49.

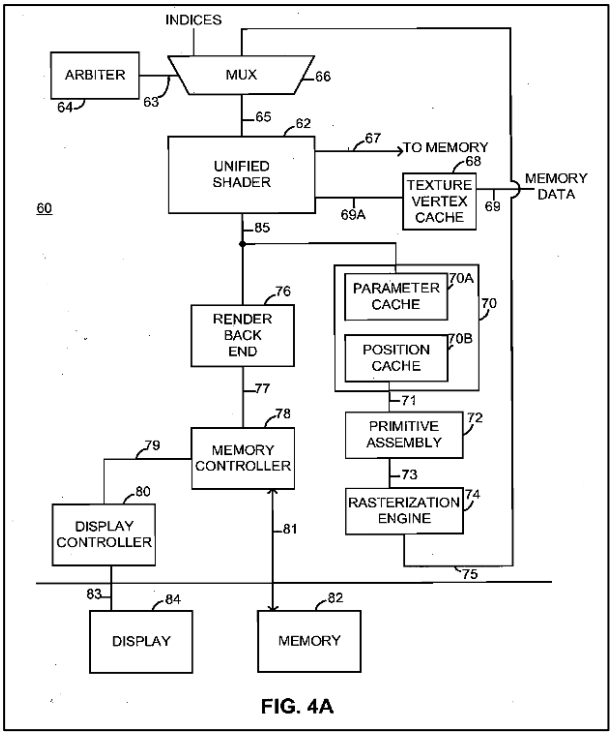


FIG. 4A

See EX. 2080 ('871 Patent) at Figure 4A.

b. "a processor unit operative to perform vertex calculation operations and pixel calculation operations"

269. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a processor.

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

See EX. 2080 ('871 Patent) at 4:9-20.

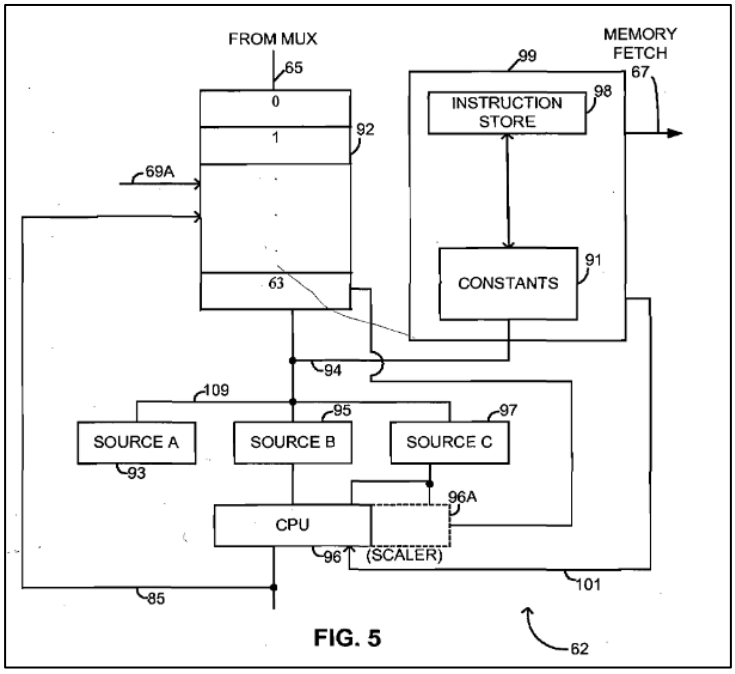


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

270. The processor can perform floating point arithmetic operations as well as logical operations on corresponding operands. The processor is logically partitioned into two sections including a portion able to perform floating point operations and a portion capable of performing scalar operations.

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

See EX. 2080 (‘871 Patent) at 4:21-32.

271. Further, the processor of the unified shader can perform both vertex and pixel operations. Specifically, the processor of the unified shader has the ability to simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

> As discussed above, the instruction store **98** maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader **99** of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader **62** of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.
>
> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 (‘871 Patent) at 5:1-20.

c. “shared resources, operatively coupled to the processor unit;”

272. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a general purpose register block and a plurality of source registers that are operatively coupled to the processor of the unified shader.

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

See EX. 2080 ('871 Patent) at 4:9-20.

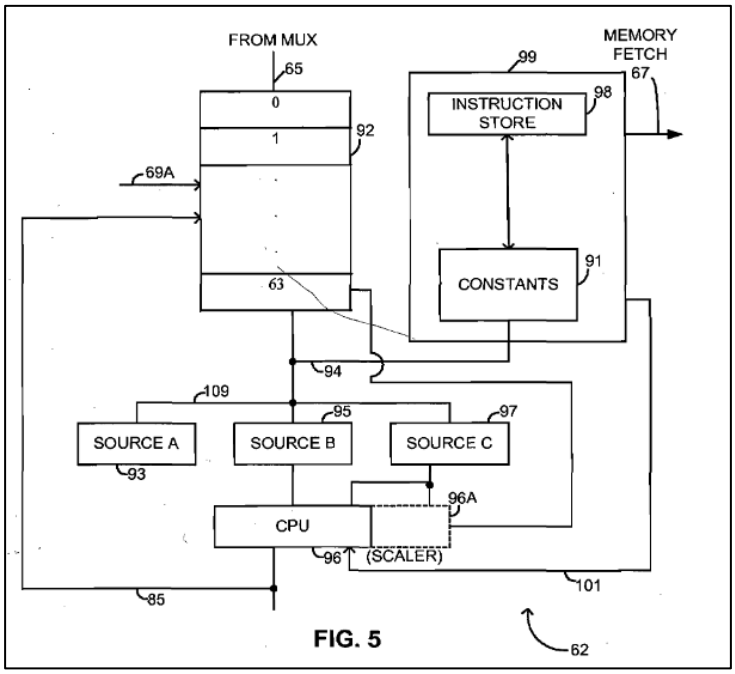


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

273. The general purpose register block can store vertex or pixel data as it receives pixel or vertex data from the multiplexer.

a sequencer **99**. The general purpose register block **92** includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer **66** on line **65** or any other information to be maintained within the unified shader. The data present in the general purpose register block **92** is transmitted to the plurality of source registers via line **109**.

See EX. 2080 ('871 Patent) at 4:14-20.

274. The multiplexer, itself, receives vertex data or pixel data.

Referring now to FIG. **4A**, in an exemplary embodiment, the graphics processor **60** of the present invention includes a multiplexer **66** having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine **74** provided at a second input. A control signal generated by

See EX. 2080 ('871 Patent) at 3:60-65.

275. If the general purpose register block does not have enough available space to store incoming vertex data, the vertex data will not be transmitted and pixel calculation operations will either be continued or started. Through the sharing of resources within the unified shader, processing of image data is enhanced as there is no down time associated with the processor.

> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**. The instruction store **98** then passes the corresponding control signals to the processor **96** on line **01** to perform such vertex operations. However, if the general purpose register block **92** does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter **64** is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor **96** are continued, based on the instructions maintained in the instruction store **98**, until enough registers within the general purpose register block **92** become available. Thus, through the sharing of resources within the unified shader **62**, processing of image data is enhanced as there is no down time associated with the processor **96**.

See EX. 2080 ('871 Patent) at 5:14-34.

d. "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."

276. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a processor that can perform floating point arithmetic operations as well as logical operations on corresponding operands.

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

See EX. 2080 ('871 Patent) at 4:21-32.

277. Further, the processor unit uses operands from the general purpose register block (into the source registers) to execute instructions. The result of executing the operation may be transmitted back to the general purpose register block.

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

See EX. 2080 ('871 Patent) at 4:21-32.

278. As described previously, the processor of the unified shader can perform both vertex and pixel operations. Specifically, the processor of the unified shader has the ability to simultaneously perform vertex and pixel operations at

various degrees of completion by being able to freely switch between such programs or instructions.

> As discussed above, the instruction store **98** maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader **99** of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader **62** of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.
>
> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 ('871 Patent) at 5:1-20.

279. Moreover, the processor in the unified shader will perform pixel calculation operations until enough registers within the general purpose register block become available. Then, the unified shader will perform vertex calculation operations.

> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**. The instruction store **98** then passes the corresponding control signals to the processor **96** on line **01** to perform such vertex operations. However, if the general purpose register block **92** does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter **64** is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor **96** are continued, based on the instructions maintained in the instruction store **98**, until enough registers within the general purpose register block **92** become available. Thus, through the sharing of resources within the unified shader **62**, processing of image data is enhanced as there is no down time associated with the processor **96**.

See EX. 2080 ('871 Patent) at 5:14-34.

280. For example, the arbiter employs an arbitration scheme as follows: the vertex data on the first input of the multiplexer is transmitted to the unified shader if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader for further processing.

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

See EX. 2080 ('871 Patent) at 3:60-4:8.

5. Claim 5

a. "A unified shader"

281. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of performing vertex and pixel operation in an efficient manner.

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

See EX. 2080 ('871 Patent) at 2:36-49.

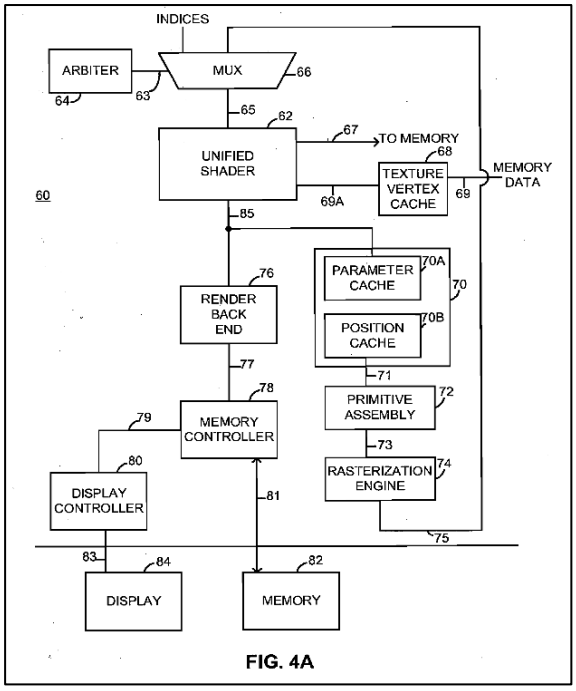


FIG. 4A

See EX. 2080 ('871 Patent) at Figure 4A.

b. "a processor unit"

282. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a processor.

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

See EX. 2080 ('871 Patent) at 4:9-20.

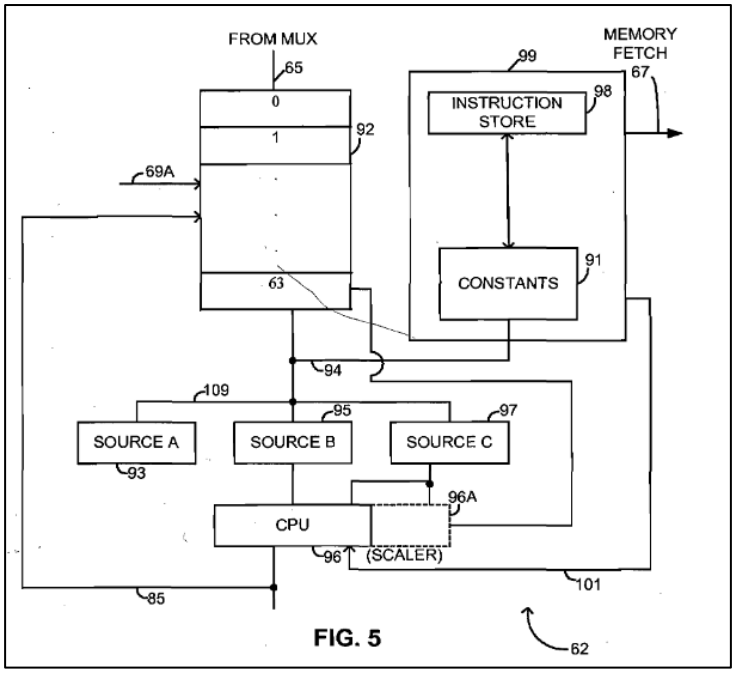


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

c. "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."

283. The 13/109,738 application provides a written description for this element. For example, the application provides that the shader includes a general purpose register block for storing the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations, and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs according to the instruction maintained in the sequencer.

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

See EX. 2080 ('871 Patent) at 2:50-63.

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

See EX. 2080 ('871 Patent) at 4:9-20.

284. The sequencer, general purpose register, and processor unit are coupled.

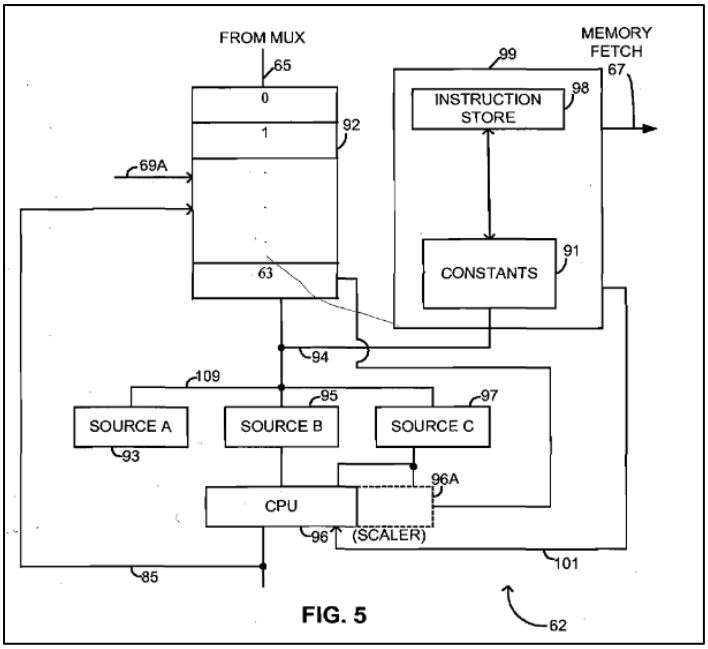


FIG. 5

See EX. 2080 (‘871 Patent) at Figure 5.

285. The sequencer maintains instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block. For example, the sequencer includes an instruction store that maintains the instructions necessary for the processor to execute arithmetic and logic operations on the data maintained in the general purpose register block. As previously described, the processor performs pixel and vertex operations.

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

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

See EX. 2080 (‘871 Patent) at 4:33-67, 2:36-63.

6. Claim 6

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

286. The 13/109,738 application provides a written description for this element. For example, the application describes that the sequencer includes an

instruction store and constant block, and the instruction store is capable of supplying memory instructions that, when executed by the unified shader, fetch texture and other types of data from memory.

> The sequencer **99** includes constants block **91** and an instruction store **98**. The constants block **91** contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store **98** contains the necessary instructions that are executed by the processor **96** in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block **92** as provided by the source registers **93–95**. The instruction store **98** further includes memory fetch instructions that, when executed, causes the unified shader **62** to fetch texture and other types of data, from memory **82** (FIG. **4**A). In operation, the sequencer **99** determines whether the next instruction to be executed (from the instruction store **98**) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer **99** sends the request to a fetch block (not shown) which retrieves the required information from memory **82** (FIG. **4**A). The retrieved information is then transmitted to the sequencer **99**, through the vertex texture cache **68** (FIG. **4**A) as described in greater detail below.
>
> If the next instruction to be executed is an arithmetic or logical instruction, the sequencer **99** causes the appropriate operands to be transferred from the general purpose register block **92** into the appropriate source registers (**93**, **95**, **97**) for execution, and an appropriate signal is sent to the processor **96** on line **101** indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor **96** executes the instructions on the operands present in the source registers and provides the result on line **85**. The information present on line **85** may be transmitted back to the general purpose register block **92** for storage, or transmitted to succeeding components of the graphics processor **60**.

See EX. 2080 (‘871 Patent) at 4:33-67, 2:36-63.

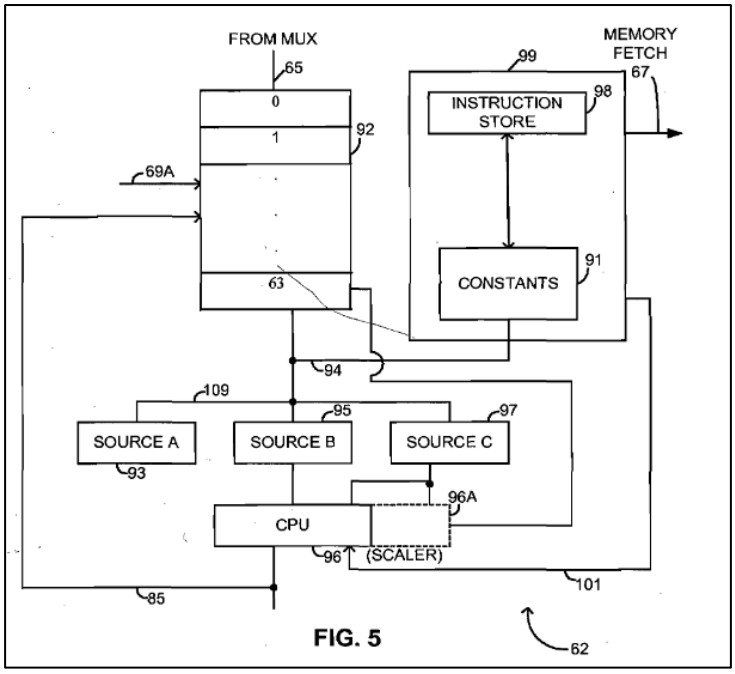


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

7. Claim 7

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

287. The 13/109,738 application provides a written description for this element. For example, the application describes that multiplexer (66) determines which of its two inputs will be transmitted to the unified shader for processing based on a control signal sent by the arbiter.

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

See EX. 2080 ('871 Patent) at 3:60-4:8.

8. Claim 8

a. "The shader of claim 5, wherein the processor unit executes instructions that generate a pixel color in response to the selected data."

288. The 13/109,738 application provides a written description for this element. For example, the processor of the unified shader can perform both vertex and pixel operations. *See* EX. 2080 ('871 Patent) at 5:1-20.

289. When the unified shader performs pixel operations it will generate color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels.

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

See EX. 2080 ('871 Patent) at 3:33-51.

9. Claim 9

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

290. The 13/109,738 application provides a written description for this element. For example, the processor of the unified shader can perform both vertex and pixel operations. *See* EX. 2080 ('871 Patent) at 5:1-20, 2:36-63.

291. When the unified shader performs vertex shading, it applies a transformation position matrix to the input position information and generates data representing a perspectively corrected image of the object to be rendered.

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

See EX. 2080 ('871 Patent) at 3:1-18.

10. Claim 10

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

292. The 13/109,738 application provides a written description for this element. For example, the application describes that multiplexer (66) determines which of its two inputs will be transmitted to the unified shader for processing based on a control signal sent by the arbiter.

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

See EX. 2080 ('871 Patent) at 3:60-4:8.

11. Claim 11

a. "A unified shader"

293. The 13/109,738 application provides a written description for this element. For example, the application describes a graphics processor that employs a unified shader capable of performing vertex and pixel operation in an efficient manner.

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

See EX. 2080 ('871 Patent) at 2:36-49.

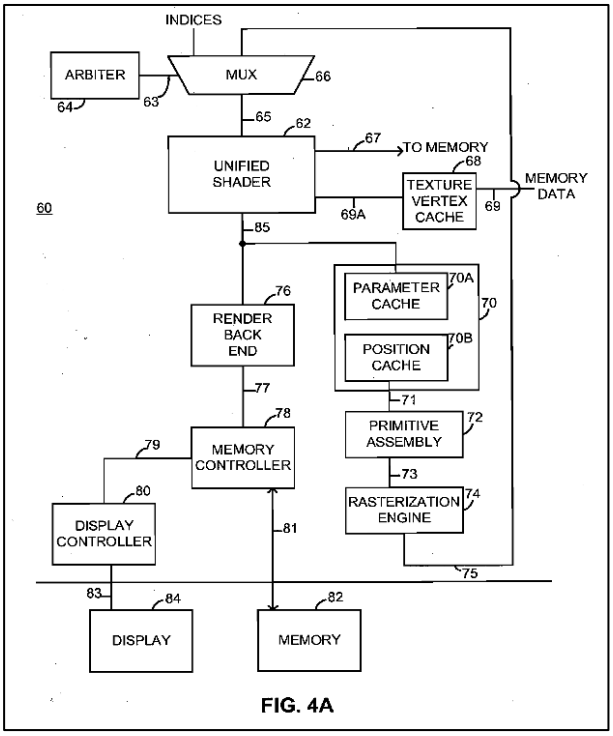


FIG. 4A

See EX. 2080 ('871 Patent) at Figure 4A.

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

294. The 13/109,738 application provides a written description for this element. For example, the application describes the unified shader as including a processor.

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

See EX. 2080 ('871 Patent) at 4:9-20.

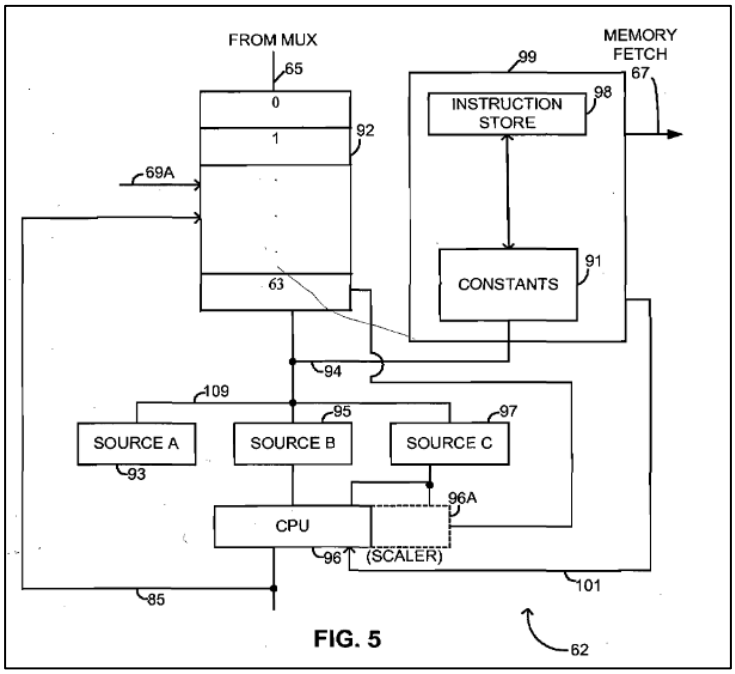


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

295. The processor of the unified shader performs both vertex and pixel operations. Specifically, the processor of the unified shader can simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

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

The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 ('871 Patent) at 5:1-20.

296. The arbitration scheme is based on pixel or vertex workload as follows: the vertex data on the first input of the multiplexer is transmitted to the unified shader if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader for further processing.

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

See EX. 2080 ('871 Patent) at 3:60-4:8.

c. "an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store."

297. The 13/109,738 application provides a written description for this element. For example, the unified shader includes an instruction store.

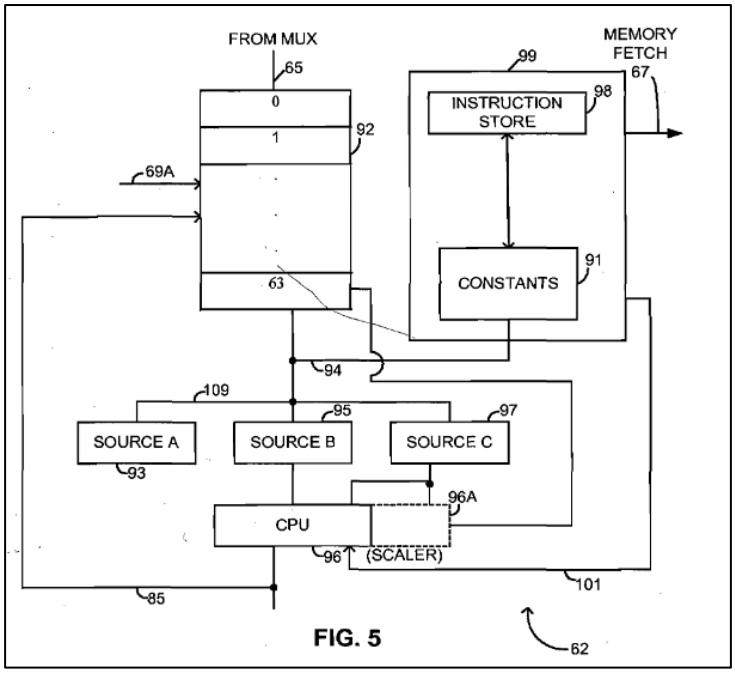


FIG. 5

See EX. 2080 ('871 Patent) at Figure 5.

298. As described previously, the processor of the unified shader can simultaneously perform vertex and pixel operations at various degrees of completion by being able to freely switch between such programs or instructions.

> As discussed above, the instruction store **98** maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader **99** of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader **62** of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.
>
> The unified shader **62** has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store **98**, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block **92** from multiplexer **66**.

See EX. 2080 (‘871 Patent) at 5:1-20.

299. The arbitration scheme is based on pixel or vertex workload as follows: the vertex data on the first input of the multiplexer is transmitted to the unified shader if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader for further processing.

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

See EX. 2080 (‘871 Patent) at 3:60-4:8.

XI. NO ABANDONMENT, SUPPRESSION, OR CONCEALMENT OF R400

300. For the same reasons that diligence was utilized during the actual reduction to practice of the R400, there was also no abandonment, concealment, or suppression up until the constructive reduction to practice of the claimed inventions on November 20, 2003. *See* Section IX, *supra*. Specifically, a team of engineers worked every non-holiday workday towards reducing the ’454 Patent to practice.

See EX. 2004 (Lefebvre Decl., IPR2015-00325) at 23, 33-58 (diligence calendar showing work on R400 documents from August 2001 to November 2003); *see also* EX. 2102 (Lefebvre Dep.) at 257:4-11; *see generally* EX. 2007 (IPR2015-0325, 326, 330 Lefebvre Dep.); EX. 2118. Engineers involved in the development of the R400 include numerous ATI employees identified in the project allocation outlines of the R400 Program Reviews. *See generally* EX. 2111 (R400 Program Reviews) at pp. 1-6, 16-52; EX. 2015, at p 4.

301. The development of the R400 eventually split into two continuous, but distinct, projects – the R500 and Xenos projects. *See* EX. 2113 (Gruber Dep.), at 210:17-211:10; *see also* EX. 2101 (Morein Dep.), at pp. 63-65. In fact, the R500 and Xenos projects represented seamless extensions of the R400 project. EX. 2113 (Gruber Dep.), at 210:17-211:10.

302. It was the Xenos project that would ultimately power the (then) next generation Xbox 360 console. *See* EX. 2102 (Lefebvre Dep.) at 132:1- 149:25, 198:1-199:25, 209:1-25, 213:1-25, 216:1-218:25, 233:2-25, 240:1-241:25, 246:1-247:25. Documents maintained in Perforce identify the continued activity of reducing the R400 (then Xenos) to practice. For example, Perforce cataloged change histories, specifications, presentations, and test plans related to the continuous diligence of the R400 project.

XII. ASSERTED PRIOR ART

A. Lindholm '685

303. Lindholm '685 regards a graphics architecture with a pixel input buffer and a separate vertex input buffer, and several execution pipelines:

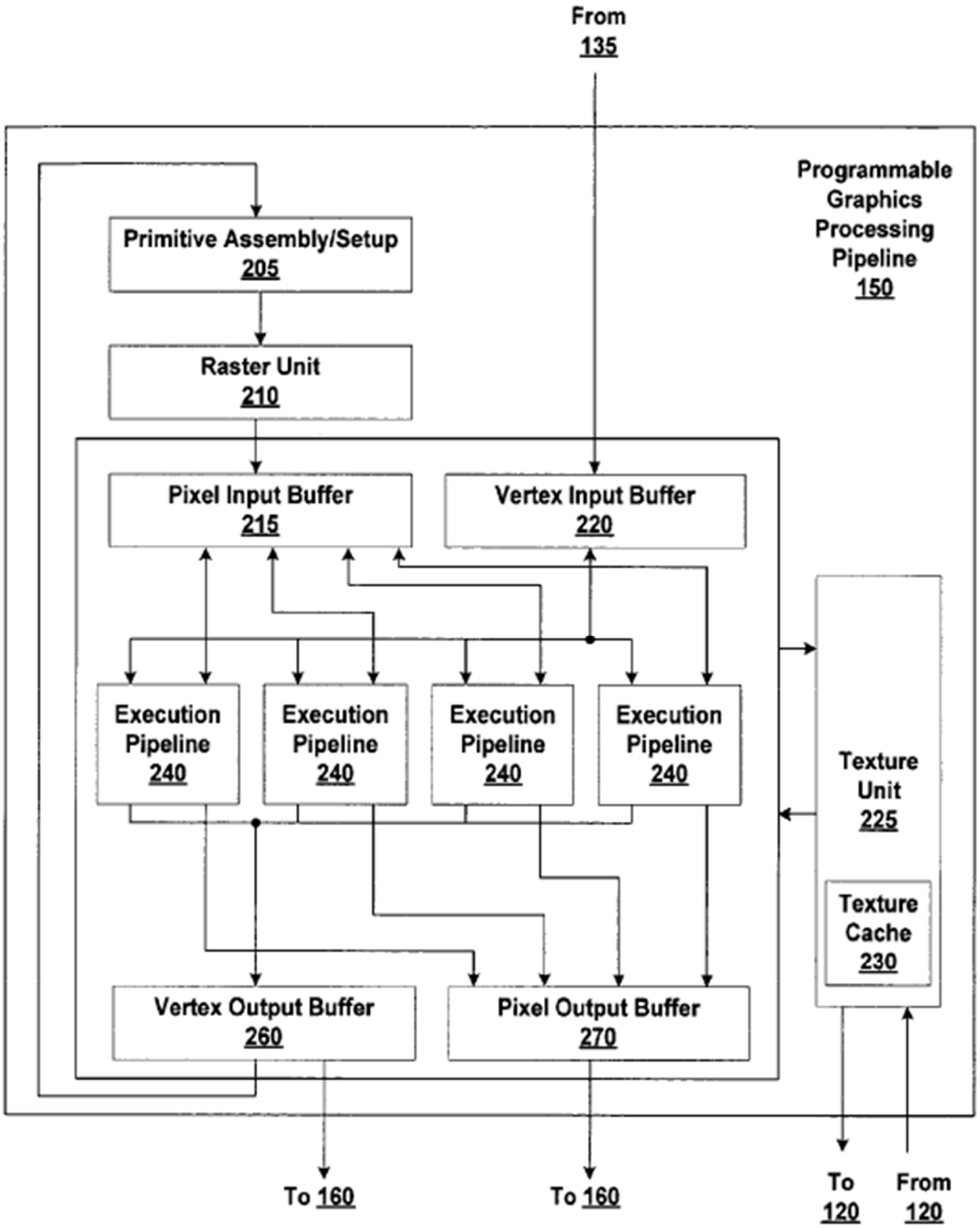


FIG. 2

EX. 1006 (Lindholm '685) at Fig. 2. The execution pipelines are each a multithreaded device:

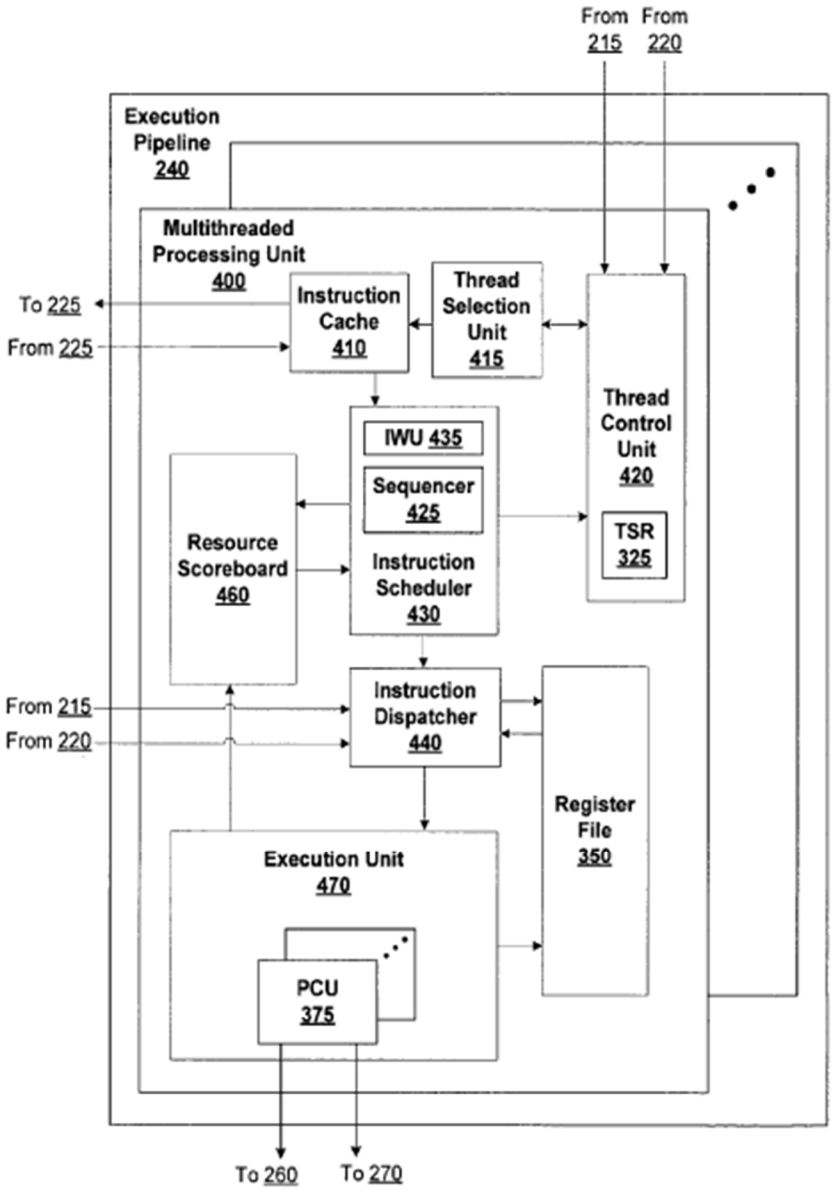


FIG. 4

Id. at Fig. 4. In response to "program instructions," "programmable computation units (PCUs) within an Execution Pipeline 240 [] perform operations such as

tessellation, perspective correction, texture mapping, shading, blending, and the like." *Id.* at 5:23-29. "Processed samples are output from each Execution Pipeline 240 to a Pixel Output Buffer 270," and then "[t]he processed samples are output from Pixel Output Buffer 270 to Raster Analyzer 160." *Id*. at 5:30-35. After rasterization, the pixel data is sent to another PCU for pixel processing. *Id*. at Figs, 2, 4, 10:55-11:14. Lindholm '685 further describes using thread allocation priorities or the amount of vertex processing to be performed to determine what processes the PCUs should perform. *Id.* at 6:60-7:5, 7:56-64.

B. Amanatides

304. Amanatides regards a graphics architecture in which a "host distributes primitives in a round-robin manner" to a variety of processors, labeled "G/R":

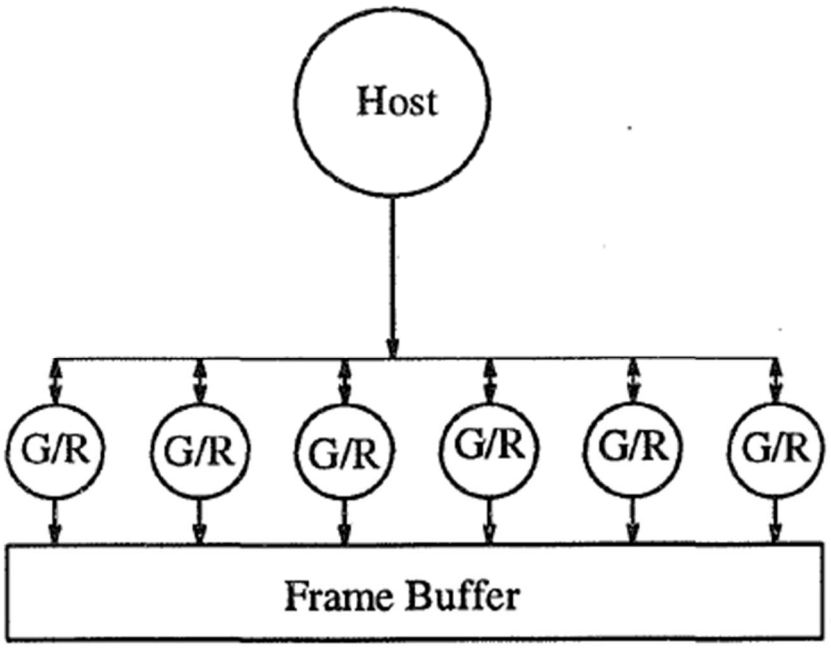


Figure 3

EX. 1007 (Amanatides) at Fig. 3, 157. "Each processor performs the front-end tasks of transforming / clipping / lighting / perspective for the primitives. It then broadcasts the results to the other processors to perform the back end operations of rasterization." *Id*. at 157. The reference also discusses "load balancing" based on "[f]or example, if there is a lot of texture mapping" tasks to be done and based on prioritization of certain types of actions. *Id*. at 157, 159.

C. Selzer

305. Selzer describes a graphics architecture with a traditional graphics pipeline involving a first stage of geometry processing, followed by a second stage of rendering processing:

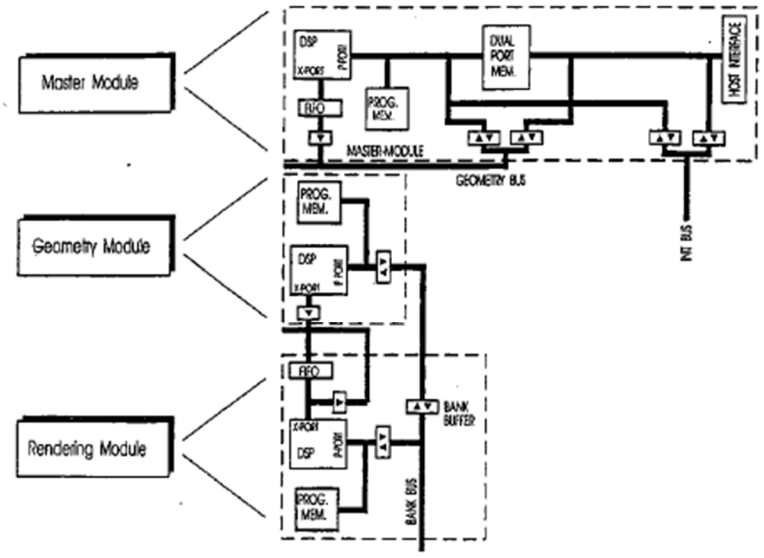


FIGURE 4.2. Basic module structure

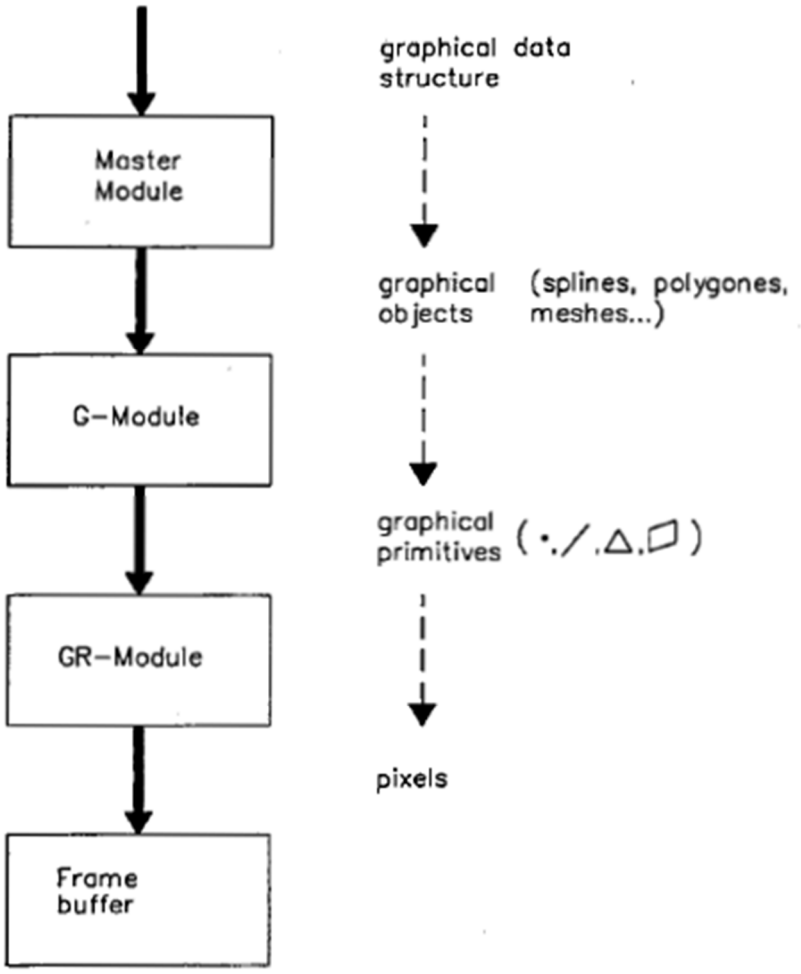


FIGURE 4.5. Graphics data processing

EX. 1009 (Selzer) at Figs. 4.2, 4.4. "The geometry module performs the transformation, clipping, polygon and patch subdivision, normal interpolation and renormalisation and lighting operations." *Id*. at 41. Alternatively, a rendering module could perform that task. *Id*. After that step, the geometry module or rendering module that performed the geometry processing "deliver[s] processed data to the FIFOs of the appropriate [separate] rendering modules" for rendering processing. *Id*.

XIII. EVALUATION OF PETITIONER'S PROPOSED GROUNDS

A. Ground 1: Lindholm '685 and Lindholm '913 Do Not Render the '454 Patent Obvious

306. In my opinion, the Lindholm patents are not prior art to the inventions of the '454 Patent. *See* Sections VII, VIII, IX, X, and XI, *supra*.

307. For at least the reasons below, the Lindholm patents also do not render the claims of the '454 Patent obvious.

1. No Executing Operations Depending Upon an Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4)

308. Claims 1 and 3-10 require consideration of the amount of available space for storage impacting processing or data selection, as explained above in Section V.B.

Claim 1 reads:

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

EX. 1001, claim 1.

Claim 3 reads:

> A unified shader comprising:
>
> a processor unit operative to perform vertex calculation operations and pixel calculation operations; and
>
> shared resources, operatively coupled to the processor unit;
>
> the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

EX. 1001, claim 3.

Claim 4 reads:

> A unified shader comprising:
>
> a processor unit operative to perform vertex calculation operations and pixel calculation operations; and
>
> shared resources, operatively coupled to the processor unit;
>
> the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

EX. 1001, claim 4.

Claim 5 reads:

> A unified shader comprising:
>
> a processor unit;

ATI Ex. No 2121
IPR2023-00922
Page 228 of 324

> 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.

EX. 1001, claim 5.

Neither Lindholm reference discusses considering the amount of space available, let alone that consideration impacting processing of particular data.

309. Petitioner's expert, Dr. Pfister, identifies Register File 350, and Pixel Input Buffer 215 and Vertex Input Buffer 220, as the "general purpose register block," "shared resources," and/or "store" of the '454 patent claims. EX. 1003, ¶¶92-94, 120. Dr. Pfister's opinion as to this claim requirement are provided only in the context of claim 3. EX. 1003, ¶¶122-128, 140 (no substantive allegations for claims 4, 5, or 1).

310. Dr. Pfister relies on several different disclosures from Lindholm which say the same thing: priority based on the amount of work of a given type discloses this claim limitation. EX. 1003, ¶108 ("giving priority to allocating vertex threads and pixel threads are only allocated if vertex samples are not available for assignment to a thread"), ¶109 (priority "according to the number of pixel samples queued for processing"), ¶110 (prioritization based on "vertex or pixel workload, depending on which is greater"), ¶109 ("a dynamic priority can be employed to flexibly direct the Execution Units/PCUs to execute vertex or pixel operations

depending on the vertex or pixel workload"), ¶109 (priority based on "a number of 'pending' pixels"). None of these consider the amount of storage, much less the amount of storage for a given type of data, or any of the other more specific requirements regarding what to process based on storage space, as described above in Section V.B.

311. The statement in Lindholm '685 that "[e]ach 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" also does not disclose the claims. EX. 1005, 8:49-53. Rather, the quote explains that the analysis concerns only whether processing resources are free ("the available of computation resources in an Execution Unit 470") or whether the source data is present such that it is capable of being processed ("source data stored in Register File 350")—it does not concern evaluating available storage space, much less doing so to determine what to execute.

312. Nor does Dr. Pfister's opinion that "[t]hread 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'" assess storage space. EX. 1003, ¶111 (citing EX. 1005, 12:11-13). First, that is a mischaracterization, because the reference does not say thread allocation decisions can be made based on that evaluation of space. Rather, all it says is the common-

sense statement that "*[t]he maximum number of threads that can be executed* simultaneously is related to the number of Execution Pipelines 240, the size of storage for thread state data, the amount of storage for intermediate data generated during processing of a sample" — *i.e.*, if there is more space, more threads can be executed. The truism that how many threads can be executed is tied to space does nothing to disclose the required evaluation of available storage space to determine what to execute at a given time, *e.g.*, in claim 1, "performing vertex operations on the vertex data by a processor within the unified shader unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available." Second, storage capability for intermediate data is not the proper question — it is storage for incoming vertex data (to be operated on) that is recited, for example, in claim 1.

313. Further, I disagree that Lindholm '685 discloses determining thread execution priority based on the amount of sample data in the pixel and vertex input buffers and recognizes that the maximum number of threads that can be executed simultaneously depends on the size of storage for thread state data. As explained above, neither discloses the required considerations for determining what to process.

For example, knowing how much data is in the buffers, and thus what to prioritize because there is more of it, does not disclose the requirement of claim 1 to perform vertex operations until there is no space for new incoming vertex data, in which case performing only pixel calculations until there is once again room for new vertex data. Nor does it disclose any consideration of space in determining which data type to execute, as also required by claims 3, 4, and 5.

2. No Execution of Instructions "In Response to" Receiving Selected Data (Claim 2)

314. Claim 2 requires a unified shader processor that executes certain pixel or vertex thread instructions "in response to" receiving selected data from a data storage, as explained above in Section V.D.

Claim 2 reads:

> A unified shader, comprising:
>
> a general purpose register block for maintaining data;
>
> a processor unit;
>
> a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block; and
>
> wherein the processor unit executes instructions that generate a pixel color in response to selected data from the general purpose register block and generates vertex position and appearance data in response to selected data from the general purpose register block.

EX. 1001, Claim 2.

The Lindholm patents do not disclose claim 2.

315. Dr. Pfister opines that the "PCU" in Lindholm '685 execute instructions that generate a pixel color in response to selected data. EX. 1003, ¶97 (citing EX. 1005, 5:23-35). This citation provides that "[p]rogram instructions configure programmable computation units (PCUs) within an Execution Pipeline 240 to perform operations," which does not indicate that receipt of data causes execution of instructions. EX. 1005, 5:23-35. The subsequent passage also confirms that the receipt of program instructions, not the receipt of data, causes execution by the PCUs. It provides: "PCUs 375 are configured using program instructions. . . . PCUs 375 performs computations specified by the program instructions." *Id.*, 5:36-49. Moreover, Dr. Pfister points to "thread allocation" to processors, which indicates that threads, not selected data, cause execution of processes. *See, e.g.*, EX. 1003, ¶¶75-76, 100, 107-108; EX. 1005, 9:5-7. Thus, it is the allocation of threads, not the receipt of data, which causes execution of instructions.

316. The embodiments in which locations in Register File 350 are assigned to threads based on priority assigned to each sample type also do not disclose the receipt of data as causing the execution of instructions. *See* EX. 1005, 6:60-7:36. These disclosures provide that threads are assigned to source samples or allocated space in Register File 350. *Id.* The purpose of allocating space in Register File 350

is for "retain[ing] intermediate data generated during execution of program instructions associated with the thread." EX. 1005, 7:37-40. Thus, the space allocated in Register File 350 is for data created after the thread is processed, and not for data that is to be processed, which is what claim 2 requires. EX. 1001, Claim 2 ("the processor unit executes instructions . . . *in response to* selected data from the general purpose register block").

3. No "Memory" Separate From the "Store" (Claim 6)

317. Claim 6, which is dependent on claim 5, requires "the sequencer further includes circuitry operative to fetch data from a memory." EX. 1001, Claim 6. Based on the use of two different terms within the same claim, a POSITA would understand that the claimed "memory" of claim 6 must be different from the claimed "store" of claim 5.

318. Dr. Pfister identifies "Instruction Dispatcher 440" as the alleged circuitry. EX. 1003, ¶129. But Dr. Pfister identifies the same components for the "memory" as it does for the "store" of claim 5, Pixel Input Buffer, Vertex Input Buffer, and Register File 350, by claiming that Instruction Dispatcher gathers source data from them. Compare *id.* (opining "fetch data from a memory" is met by "gather[ing] the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350") *with* EX. 1003, ¶128 (referencing claim elements 2[c], 3[a] and [c], ¶106 (identifying "store" as "*i.e.*, the Register File 350 and Vertex/Pixel

Input Buffers (claim 5)").

319. Dr. Pfister has not explained that these are alternate theories – or alleged that in the instance that Register File 350 is the "store," Pixel Input Buffer and Vertex Input Buffer are the "memory," or vice versa. EX. 1003, ¶129.

4. No "Control Signal" (Claim 7) or "Arbiter" (Claim 10)

320. Claim 7 requires "control signal," and claim 10 requires an "arbiter," which provides the "control signal" of claim 7. EX. 1001, Claims 7, 10. Specifically, claim 7 reads:

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

EX. 1001, claim 7.

Claim 10 reads:

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

EX. 1001, claim 10.

321. Dr. Pfister identifies Thread Control Unit 420 as the claimed "arbiter." EX. 1003, ¶131. According to Dr. Pfister, Thread Control Unit "allocates threads or samples to threads based on the thread allocation priority or execution priority." *Id.* But a POSITA would understand that priority information is not equivalent to control signals, which are "electrical ***signals*** used to control internal or external devices or processes." EX. 2126, 128 (Microsoft Computer Dictionary) (emphasis added). This definition is consistent with how the '454 patent's specification

describes signals: "[A]n appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers." EX. 1001, 5:10-13.

322. Instead, a POSITA would understand that the priority information identified by Dr. Pfister is data, and not control signals. Dr. Pfister explains that Thread Selection Unit 415, the alleged selection circuit, "reads ***thread entries*** (including thread state data) from the Thread Control Unit." EX. 1003, ¶132 (emphasis added). The thread entries and thread state data being read from the alleged arbiter are data, as Dr. Pfister admits: "Thread Selection Unit outputs *thread entries* and other thread data, such as state data, which provides *information* to the 'store'." *Id.* (emphasis added).

323. Finally, Dr. Pfister has not explained how the alleged control signals are "provided by" the alleged arbiter. Dr. Pfister's alleged selection circuit, "***reads*** thread entries (including thread state data) from the Thread Control Unit." EX. 1003, ¶131 (emphasis added). This does not meet the limitation of the arbiter providing a control signal, rather describes the alleged selection circuit reading information from the alleged arbiter.

5. No Processor Unit That Performs Vertex Manipulation Operations and Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11)

324. Claim 11 requires that "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." EX. 1001, Claim 11. Specifically, claim 11 reads:

> A unified shader comprising:
>
> > a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload; and
> >
> > an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

EX. 1001, claim 11. At a minimum, this claim requires the processor unit to switch between vertex operations and pixel operations at various degrees of completion.

325. Dr. Pfister does not provide any independent analysis for this claim, instead relying on sections from claims 2[c], [d], 3[a], [c]. EX. 1003, ¶¶137-138.

326. Dr. Pfister first cites to EX. 1005, 12:9-27 and EX. 1006, 7:6-12 for the disclosure of this requirement. EX. 1003, ¶113. These sections are duplicates of each other. The cited to portions of the Lindholm patents provide that if there are not enough threads available, then not all samples within a set of one type can be processed simultaneously. EX. 1005, 12:14-20. If there enough threads available, then more than one set of samples of one type can be processed simultaneously. *Id.*, 20-23. If not enough threads are available, they can be allocated, for example, from

Portion 620 to 630. *Id.*, 24-29. Portions 620 and 630 store thread state data. *Id.*, 11:42-45. None of these disclosures explain or disclose that Lindholm can switch between unfinished vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store. Nor does Dr. Pfister explain these disclosures do so; his opinion only characterizes the disclosures as "explain[ing] how the distribution of processing resources between vertex and pixel operations can be altered during operation," without any further explanation. EX. 1003, ¶113.

327. Dr. Pfister also opines that instruction-level scheduling and the ability to "concurrently process multiple samples" discloses claim 11. EX. 1003, ¶¶114-118. However, these disclosures do not discuss switching between different sample types, nor even switching between different samples. The concurrent processing of different samples does not indicate that any samples are paused or switched such that they are at "various degrees of completion" as required by claim 11.

328. First, Dr. Pfister points to the issuance of instructions: When Instruction Scheduler 430 determines which instructions and associated threads will be executed, Instruction Scheduler 430 outputs at least one instruction to a dispatcher. EX. 1003, ¶116 (citing EX. 1006, 9:24-30). The issuance of instructions does not disclose "performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions

in the instruction store." There is no disclosure that the issued instructions must be both pixel and vertex instructions, nor that the instructions cannot pertain to the same operation, or that the issued instructions are when the operations are at varying degrees of completion.

329. Next Dr. Pfister discusses the ability to simultaneously process multiple samples. This ability merely provides that multiple threads are processed at the same time. It also does not disclose anything regarding: (i) whether the threads are both pixel and vertex operations; (ii) whether the threads are at varying degrees of completion; or (iii) whether switching instructions occurs. For example, the threads could have the same instructions but be acting on different data.

B. Ground 2: Amanatides and Kohn Do Not Render the '454 Patent Obvious

330. In my opinion, Ground 2, which is based on Amanatides and Kohn, does not render obvious the claims of the '454 Patent.

1. No Executing Operations Depending Upon An Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4)

331. As explained above in Section V.B, the claims of the '454 patent recites three levels of specificity in terms of ways the claimed invention must evaluate space available in a data storage to determine what operations to perform.

Claim 1 reads:

A method carried out by a unified shader comprising:

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

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

EX. 1001, claim 1.

Claim 3 reads:

A unified shader comprising:

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

shared resources, operatively coupled to the processor unit;

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

EX. 1001, claim 3.

Claim 4 reads:

A unified shader comprising:

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

shared resources, operatively coupled to the processor unit;

ATI Ex. No 2121
IPR2023-00922
Page 240 of 324

> 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.

EX. 1001, claim 4.

Claim 5 reads:

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

EX. 1001, claim 5.

Ground 2 does not disclose any.

332. Dr. Pfister first points to Amanatides' "load balancing," which states "[f]or 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." EX. 1007, 157; EX. 1003, ¶183. Dr. Pfister concludes this "disclosure encompasses at least the load balancing as stated in claims 5 and 11, which involves conducting a greater number of vertex operations or pixel operations depending on the workload of vertex/pixel data . . . or *the available memory for each type of task available*." EX. 1003, ¶183. But this is

incorrect on its face. The citation by Dr. Pfister merely discloses, as it says, if there is a lot of one type of work to do, the processors spend more time on that type of work. There is no disclosure of considering how much *available* memory space there is for each type of work.

333. Dr. Pfister then points to Amanatides' high and low priority packets. EX. 1003, ¶184. The high- and low-priority designations are a bus-sharing protocol, in which "[l]ow priority packets get onto the bus only if no high-priority packets are waiting to be sent." EX. 1007, 159. As with the above, this passage does not disclose considering the amount of available storage space.

334. Amanatides' priority scheme also does not disclose the above elements. In Amanatides there are only ever contains 3 decision points regarding what data to transmit and process, as explained below. None of them involve basing the decision of what type of packet to send for execution even in part on how much space is available in the input FIFOs, the alleged "general purpose register" (claim 1), "shared resources" (claims 3, 4), and "store" (claim 5)). EX. 1003, ¶170. Dr. Pfister also opines a few other items as comprising the general purpose register, but notably does not opine that the output FIFO is a part of the claimed register block, although that would not change the analysis. *See id.*

335. ***First***, in this scheme, GMs (the alleged vertex data and the messages that correspond to vertex operations) are sent for processing. EX. 1007, 160. This

first decision point (to the extent any decision is made at all) does not involve a determination of available space in the input FIFOs, or any other alleged component that Dr. Pfister alleges comprise the general-purpose register block, shared resources, or store. This is simply the default starting rule.

336. ***Second***, "[a]s soon as one of these GMs turns into a RM it shuts down further introduction of GMs from the host," triggering a second phase of operation. EX. 1007, 160. Thus, the second decision point, the transition from sending GMs for processing to sending only RMs for processing does not consider available storage space in the input FIFO or any other alleged component that Dr. Pfister alleges comprise the general-purpose register block, shared resources, or store. *Id.* The only consideration is that one GM has been turned into an RM. *Id.* Only that factor dictates what data type is sent and operated on in this second phase of operation. After that triggering event, in this second phase of operation, only RMs are sent to input FIFOs for processing (from the output FIFOs). EX. 1007, 159-160. Therefore, the statement that "[w]henever space appears at any input FIFO it is the RMs that are delivered to it" does not indicate an evaluation of available storage space in determining which data type to process because there is no determination to be done in this phase: only RMs are sent and processed.

337. ***Third***, a third phase of operation, in which "the host begin to deliver GMs again" is triggered by the system running out of RMs to deliver to input FIFOs

for operation, *i.e.*, "when all the output FIFOs are empty and the processors are working on the remaining RMs in their input FIFOs." *Id*., 160. This third decision point, the transition from sending only RMs for processing to sending GMs for processing, therefore also does not consider available storage space in the input FIFO or any other alleged component that Dr. Pfister alleges comprise the general-purpose register block, shared resources, or store.

338. This third transition point evaluates whether there is RM work to do, and never determines, for example in claim 1, the general-purpose register block "does not have enough available space therein to store incoming vertex data." Additionally, the output FIFO is not part of the alleged "general purpose register" (claim 1), "shared resources" (claims 3, 4), and "store" (claim 5). *E.g.*, EX. 1003, ¶170, 186. The input FIFO is alleged to fill that role, among other components, such as DRAM, a bus, and an on-chip cache. *Id.* Furthermore, the output FIFO cannot fill the role of the general-purpose register block because the general purpose register block in the claims is where data is put for processing (which Dr. Pfister contends is the input FIFO), whereas the output FIFO is the place for data after processing. EX. 1007, 159 (explaining that after GMs are processed, they are put into the output FIFO)

339. Thus, there is no disclosure in Amanatides that meets even the broadest recitation of the concept, which is in claim 5, which requires "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."

340. Amanatides further fails to disclose the narrower requirements of claims 1, 3, and 4 as well. As to claim 1, there is no consideration in Amanatides of whether the alleged "general purpose register block" (the input FIFO or other components) has enough available space to store the alleged incoming vertex data (GMs). In the first phase of operation, GMs are sent, stored and operated on; and nothing in Amanatides causes the system not to send and process GMs if "the general purpose register block does not have enough available space therein to store incoming vertex data." EX. 1007, 160. In the second phase, there very well might be space to store GMs in the input FIFOs. *Id.* But because the output FIFOs still have RMs, the system refuses to attempt to send, store, and process any GMs. *Id.* It chooses not to store GMs *not* because it cannot fit them in memory, as required by the plain claim language, but because it prefers to work on RMs until all RMs are gone to prevent deadlock. *Id.* And there is no disclosure of processing RMs "until enough registers within the general purpose register block become available," as required in claim 1. *Id.* There is simply no evaluation of whether to cease processing RMs, or begin sending and processing GMs, on the basis that there is once again enough space for GMs in the input buffers or anywhere else.

341. Similarly, as to claim 3, the system in Amanatides does not perform the

alleged pixel calculations (RMs) "until enough shared resources become available" for the alleged vertex calculations (GMs). Rather, it performs RM processing until it runs out of RMs to process. EX. 1007, 160. To the extent the space of anything is evaluated, it is the output FIFO, which are not the alleged "shared resources," which are the input FIFO. However, even that is not an evaluation of whether enough shared resources are available: it simply states that there is no more RMs to process. Enough resources may have been available before that time, or they still may not be available when there are no more RMs to process. Amanatides simply does not check.

342. Nor does Amanatides disclose claim 4. Amanatides does not perform alleged vertex processing (on GMs) until enough shared resources for the alleged pixel processing (on RMs) becomes available as required by claim 4, which requires the reverse sort of operation of claim 3. The only transition from GM processing to RM processing is explicitly "[a]s soon as one of these GMs turns into a RM." EX. 1007, 160. As explained above, the only evaluation of when to begin performing pixel processing (allegedly the RM processing), as required by claim 4 has nothing to do with storage space.

343. Claim 5 is not disclosed for an additional reason. In its analysis of claim 5, Dr. Pfister does not specifically opine what is the "store." EX. 1003, ¶¶190-192 (referring to claim elements 2[b], [c] and 3[a], [c]), ¶¶172-186 (no identification of

what the "store" is). Rather, for dependent claims 7 and 10, Pfister opines the input FIFO (and indeed all FIFOs) is a "selection circuit" that sends packets *to the store*, and thus cannot be the "store": "the 'selection circuit' stated in claim 7 is satisfied by at least the bus and FIFOs, which function to provide packets ("information") to the DRAM, VRAM, and on-chip data cache ("***store***") in response to a control signal." EX. 1003, ¶195 (emphasis added). Thus, Dr. Pfister has not alleged a theory in which the input FIFO is the "store," and so its allegations fail as to that claim for that reason alone. And none of those components alleged to be the "store," the DRAM, VRAM, and on-chip data cache, are ever said to be evaluated for storage space (or anything else) to determine what data to transmit and process in Amanatides. Thus, claim 5 has not been shown to be invalid for this additional reason.

344. Kohn is not alleged to fix the deficiency with Amanatides, but rather is used for the concept of switching "at various stages of completion." *E.g.*, EX. 1003, ¶183.

2. No Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11)

345. A POSITA would understand that claim 11 requires switching between partially complete (*i.e.*, unfinished) operations. Specifically, claim 11 reads:

A unified shader comprising:

> a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload; and
>
> an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

EX. 1001, claim 11. As is made plain by the claim language, the system must stop processing an operation while it is partially complete, and then by switching instructions in the instruction store, process a new operation. Thus, the mere switching of instructions when *e.g.*, beginning execution of a second thread simultaneously while already executing a first thread, does not disclose this claim element because "the vertex manipulation operations and pixel manipulation operations at various degrees of completion" must be "based on" the required switching of instructions in the instruction store.

346. Dr. Pfister does not rely on Amanatides for disclosure of this functionality, as Amanatides does not disclose it, but as explained in the previous section, rather only discloses a prioritization scheme for deciding whether to process GMs or RMs, but not partial operations. Instead, Dr. Pfister relies on Kohn. EX. 1003, ¶¶183, 201. Dr. Pfister points to Kohn's "dual-operation mode" within the "pipelined mode" as disclosing the ability to switch between vertex and pixel operations at various degrees of completion. *Id*., ¶201.

347. However, Kohn does not resolve the deficiency in Amanatides. Dr. Pfister argues that the existence of a "dual-operation mode" within the "pipelined mode" in Kohn discloses this limitation. *Id*. But Dr. Pfister has done nothing whatsoever to prove that fact other than to state "that the functional units within each i860 processor have the capacity to perform both vertex operations and pixel operations simultaneously, potentially at different stages of completion, due to switching between instructions stored in the instruction cache." *Id*. He provides no further analysis.

348. Instead, a POSITA would understand a pipeline does not perform operations at various stages of completion based on switching between instructions in the instruction store. In a pipeline, data comes in, one operation is done on it, then it moves to the next stage where another operation is done on it, etc. For example, Kohn describes distinct operations as the parts of its pipeline: "The RISC core contains a pipeline consisting of four stages: fetch, decode, execute, and write." EX. 1008, 17. These are distinct operations and there is no support for the argument that Kohn teaches operating them at various degrees of completion based on switching instructions. Thus, Dr. Pfister's allegation that "Kohn further highlights that both the floating-point and integer processor units consisted of multiple stages, resulting in different operations being loaded into the pipeline at different stages of execution," has no support. The fact that there might be parallel paths or "dual-

operation" does not change that, because it does not change the fact that operations are not performed in a partially-complete state based on switching instructions.

349. Moreover, Dr. Pfister does not explain of why using the pipeline of Kohn would be in any way relevant or beneficial to the load balancing or deadlock solution of Amanatides, or any other part of Amanatides, or even how the proposed combination would work. Even if some concoction could be made, Dr. Pfister has not adequately explained it or why it would be obvious and motivated. To the contrary, Amanatides explicitly discloses a solution to its deadlock issues, and any change to that solution brought by adding Kohn could reinstate the deadlock problem that Amanatides worked so hard to solve.

3. No "Selected Data" (Claims 2, 5)

350. Claims 2 and 5 of the '454 Patent require selecting data from a data store, transmitting that data to the processor, and operating on that data. EX. 1001, Claims 2-5.

Claim 2 reads:

> A unified shader, comprising:
>
> a general purpose register block for maintaining data;
>
> a processor unit;
>
> a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel

> calculation operations on selected data maintained in the general purpose register block; and
>
> wherein the processor unit executes instructions that generate a pixel color in response to selected data from the general purpose register block and generates vertex position and appearance data in response to selected data from the general purpose register block.

EX. 1001, Claim 2.

Claim 5 reads:

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

EX. 1001, claim 5.

For example, the specification explains that after one of a plurality of inputs is selected, and the shader processes that selected input. *Id.*, 2:62-3:3; *see also id.*, 4:38-39, 5:6-18. There is no process in Amanatides or Kohn for selecting data from a data store.

351. The "priority" of certain data over others in Amanatides is not a means by which data is selected "from the general purpose register block" (claim 2) or from that "maintained in a store" (claim 5). Rather, the data exits the storage in a round-robin fashion from the host, which as explained below in Section XIII.B.4 cannot be

ATI Ex. No 2121
IPR2023-00922
Page 251 of 324

the claimed "general purpose register block" or "store" because it is not part of the alleged unified shader, and then the bus-sharing rules determines if the data is permitted to "get onto the bus." EX. 1007, 159.

352. There is no selection of GMs versus RMs sitting in any alleged register block or data store. Rather, as explained above, there are only three different modes of operation: at first GM only, then RM only, then GM only. The system then "oscillate[s]" back and forth between these modes. EX. 1007, 160. This does not disclose selecting GM versus RM from a register block or data store. In the GM-only phase, GMs are sent from the host to the input FIFOs. The host is not alleged to be a register block or data store, nor could it be, because, as explained below in the following section, it is outside the alleged unified shader. After that, there is no selection; the data is simply processed. And RMs are never sent from the host; instead, all RMs in the system are generated only from the act processing GMs, and all of them are delivered then to an output FIFO (which is not alleged to be part of the register block or data store, as explained in the section above) and then to an input FIFO, when the system is operating in the RM-only mode. EX. 1007, 159-160. Thus, there is no world in which GMs and RMs are sitting in a register block or data store and are selected from.

4. Alleged "Selection Circuit" Is Not Within Alleged "Unified Shader" As Required (Claims 7, 10)

353. Claims 7 and 10 require the "shader of claim 5, further including a

selection circuit." EX. 1001, Claims 7, 10; *see* Section XIII.B.4 (explaining that the preamble is limiting), *supra*. Specifically, claim 7 reads:

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

EX. 1001, claim 7.

Claim 10 reads:

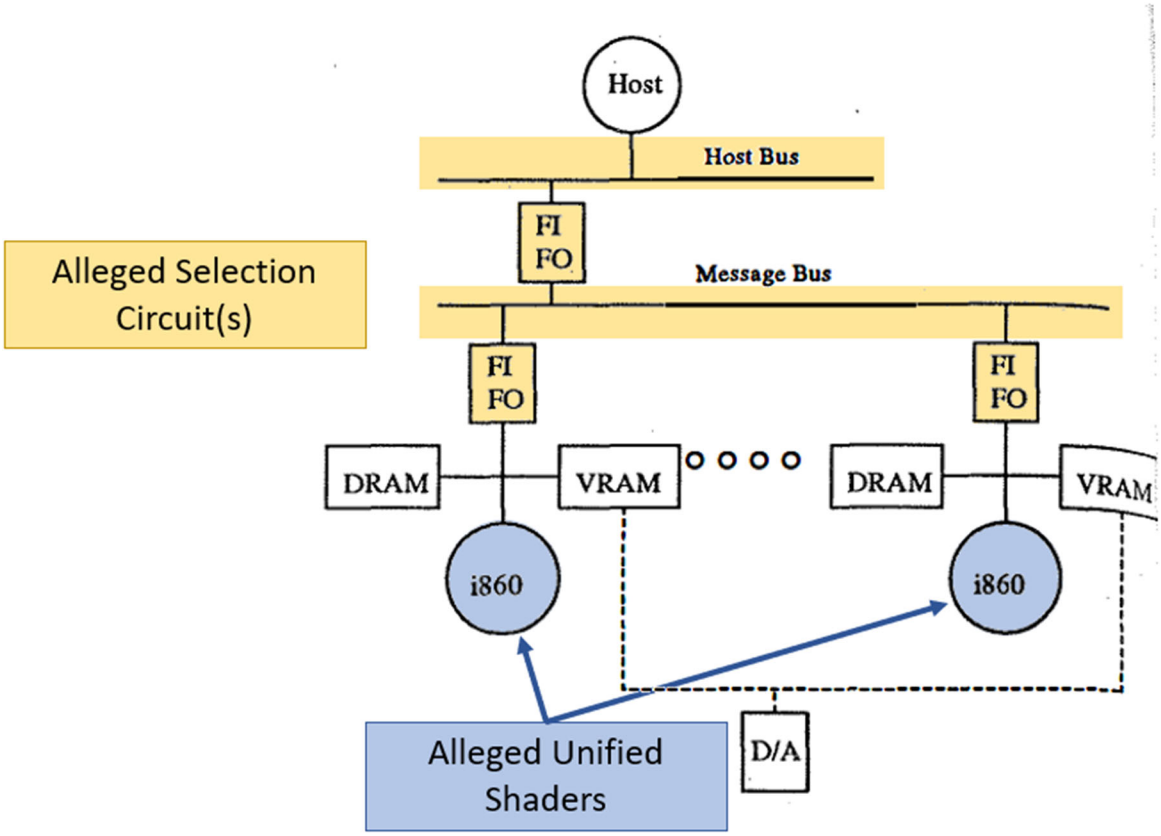
The shader of claim 7, wherein the control signal is provided by an arbiter.

EX. 1001, claim 10.

But the alleged unified shader does not include the alleged selection circuit, and thus the ground fails for claims 7 and 10.

354. Dr. Pfister alleges the unified shader is each G/R processor, which is equivalent to each i860 processor, *e.g.*, in Figure 4. EX. 1007, 158; EX. 1003, ¶169 ("Thus, Amanatides termed the unified shader unit the G/R unit."), ¶172 (referencing "each G/R illustrated in Amanatides (the i860 processor).").

355. Dr. Pfister then alleges that the selection circuit is "satisfied by at least the bus and FIFOs, which function to provide packets ('information') to the DRAM, VRAM, and on-chip data cache ('store')." EX. 1003, ¶195. But the bus and FIFOs are not part of the alleged unified shader. Figure 4 of Amanatides shows that the bus and FIFO are outside of the i860 processors.



EX. 1007, Fig. 4 (annotated). The text of Amanatides explains this is also the case: "The processors would be connected to each other and the host via a bidirectional FIFO connected to a message passing bus." *Id.* Thus, while the bus and FIFOs are used to connect the processors, they are not part of each G/R processor alleged to be the unified shader.

356. Each of the preambles recite a unified shader. EX. 1001, Claims 1-11. A POSITA would understand that all the preambles are limiting, and thus the structure recited in them is required. This is because the "unified shader" language gives additional structure which has been underscored as important by the

specification. The specification explains that the unified shader is, for example, "capable of performing both the vertex operations and the pixel operations in a space saving and computationally efficient manner." EX. 1001, 2:58-61. For example, claims 1 and 11 recite the unified shader not only in the preamble but in the body of the claims, referring to "the unified shader," making it unambiguous that the term "a unified shader," which provides antecedent basis, in preamble is limiting. The same should be true of other claims, as there is no reason to differentiate them.

357. Additionally, the applicant distinguished prior art (Shen) as having "separate vertex shader and separate pixel shader[s]," arguing there was thus "no teaching or suggestion of any unified shader." EX. 1002, 358-359. Thus, the applicant clearly relied on the preamble during prosecution to distinguish the claimed invention from the prior art.

5. No "Control Signal" (Claims 7, 10)

358. Claims 7 and 10 require a "control signal." EX. 1001, Claims 7, 10. Dr. Pfister claims the "control signal" is part of a packet's "high" or "low" priority designation. EX. 1003, ¶196. But a POSITA would not understand Amanatides's packets include control signals. As Amanatides explains, the packets are either GM or RM ***data***. EX. 1007, 159 ("Each processor waits for packets in its input FIFO. Packets containing polygons come in two flavors: geometry messages (GM) and rendering messages (RM)."). This is consistent with a POSITA's understanding of

packets, which are units of information or data being transmitted. EX. 2125, 244 (Dictionary of Computing).

359. Further, there is no disclosure that the packets themselves contain a priority designation. A POSITA would understand that given this pre-set priority scheme, it would be a waste of time to list the priority in the packet. Rather, the system would simply assign a priority based on packet type.

360. In contrast, a POSITA would understand control signals are "electrical ***signals*** used to control internal or external devices or processes." EX. 2126, 128 (Microsoft Computer Dictionary) (emphasis added); *see also* EX. 2125, 82 (defining "control signals" as "electrical signals transmitted to control the actions of a circuit"). This definition is consistent with how the '454 Patent's specification describes signals: "[A]n appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers." EX. 1001, 5:10-13. As a POSITA would know, a control signal is an instruction from the sender to the recipient, not mere data that the recipient can interpret however it chooses. Amanatides has no disclosure that its packets contain control signals, nor has Dr. Pfister identified any. Therefore, the "high" or "low" priority part of the packets identified by Dr. Pfister cannot be the claimed "control signal."

6. No "Selection Circuit" or "Arbiter," "(Claim 10)

361. Claim 7 requires "a selection circuit operative to provide information to the store in response to a control signal." Claim 10 adds the requirement of an arbiter that provides the control signal to the selection circuit.

362. Dr. Pfister alleges that the selection circuit is "satisfied by at least the bus and FIFOs, which function to provide packets ('information') to the DRAM, VRAM, and on-chip data cache ('store') in response to a control signal, such as the designation of high or low priority for the packets." EX. 1003, ¶195. But neither the bus nor the FIFOs can be "selection circuits." FIFOs are merely memory, not selection circuits. And bus lines are merely transmission lines, not selection circuits. Dr. Pfister provides no analysis as to why memory or bus lines would be a selection circuit.

363. Further, Dr. Pfister alleges that the bus and FIFOs provide packets to the DRAM, VRAM and on-chip cache to meet the claim requirement that the selection circuit be "operative to provide information to the store in response to a control signal." EX. 1003, ¶195. But it provides no evidence or analysis—all Dr. Pfister alleges is that the bus is used to transmit packets. There is no indication in Amanatides that "packets" are ever sent from the FIFOs or bus to DRAM, VRAM, or on-chip cache. Moreover, there is no indication or analysis proving that the alleged "control signal" (which is alleged to be a part of the packet itself) prompts such a transmission, as required.

364. Dr. Pfister's mapping of the "arbiter" also makes no sense. Dr. Pfister admits the references of Ground 2 lack an arbiter and instead alleges that time-slot arbitration "would be performed by logic circuits." EX. 1003, ¶196. However, he has not explained how this hypothetical arbiter sends control signals as required by claim 10, especially given Dr. Pfister's theory of control signals are already present within the packets.

365. I understand Dr. Pfister explained that the theory is that the alleged selection circuits make the decision regarding what data type to transmit, and then the alleged undisclosed arbiter is used to put the packets on the bus lines in proper time slots. EX. 2123 (Pfister Dep), 83:4-8. But in this system, the alleged arbitration is separate from and happens after selection; the decision is made and then the final sorting onto the bus via timeslots happens by action of the undisclosed arbiter. At a minimum, that is the most obvious way to implement the arbiter, and without question there can be no inherency argument that it *must* be done in the opposite order, as would be required for Dr. Pfister's theory. In that hypothetical system, there is no "control signal" sent from the undisclosed arbiter to the alleged selection circuit that prompts transmission of information from the selection circuit to the store. Instead, the undisclosed arbiter merely receives packets from the selection circuit, which has already done the selection, and puts it on a bus line.

C. Ground 3: Selzer and Fiske Do Not Render the '454 Patent Obvious

366. In my opinion, Ground 3, which is based on Selzer and Fiske, does not render obvious the claims of the ’454 Patent.

1. No Executing Operations Depending Upon An Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4)

367. As explained above in Section V.B, the ’454 patent recites three levels of specificity in terms of ways the claimed invention must evaluate space available in a data storage to determine what operations to perform.

Claim 1 reads:

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

EX. 1001, claim 1.

Claim 3 reads:

> A unified shader comprising:
> a processor unit operative to perform vertex calculation operations and pixel calculation operations; and

shared resources, operatively coupled to the processor unit;

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

EX. 1001, claim 3.

Claim 4 reads:

A unified shader comprising:

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

shared resources, operatively coupled to the processor unit;

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

EX. 1001, claim 4.

Claim 5 reads:

A unified shader comprising:

a processor unit;

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

EX. 1001, claim 5.

ATI Ex. No 2121
IPR2023-00922
Page 260 of 324

Ground 3 does not disclose any of these formulations.

368. Dr. Pfister relies on the alleged ability of Selzer to "load balance" as prioritizing vertex or pixel operations based on how much work of a given type there is to do. *E.g.*, EX. 1003, ¶¶236, 241. Selzer explains it prefers that its rendering module work on rendering calculations, and switches only to geometry calculations when the rendering module's input buffer is empty, as this approach operates at "peak performance." EX. 1009, 47-48. This priority scheme does not disclose claims 1, 3, 4, or 5, as it does not evaluate available storage space in determining what to process. Furthermore, this prioritization is not load-balancing because it in fact will push the operating system to do more render operations and fewer geometry operations, resulting in an unbalanced load.

369. Dr. Pfister also points to thread prioritization discussion in Fiske. EX. 1003, ¶¶236, 242. Fiske regards a "thread prioritization" algorithm, as stated in its title. EX. 1010, 210. "Thread prioritization is a simple means of guiding the scheduling of threads on a node. Each thread has a priority that indicates the importance of the thread in the overall problem . . . Thread prioritization involves assigning a priority to all the different threads in an application, and then using this priority to make thread scheduling decisions." *Id.*, 210-211. There is no assessment of available space for storage that impacts Fiske's priority scheme as required by the claims.

370. As explained in connection with Grounds 1 and 2 above, *see* Sections XIII.A.1 and XIII.B.1**Error! Reference source not found.**, assigning priorities to threads, data, or data types, does not disclose determining what data to transmit and process based on an evaluation of current data storage capacity, as required by claims 1 and 3-10 of the '454 Patent. *See, e.g.*, EX. 1009, 46-47 (discussing the rendering module performing geometry processing once it performs all required rendering processing); EX. 1010, 211 ("Thread prioritization involves assigning a priority to all the different threads in an application, and then using this priority to make thread scheduling decisions. . . . [T]he hardware scheduler uses the priority to determine which loaded thread to execute."). Dr. Pfister asserts without any explanation that Selzer and Fiske teaches more, such as "if enough shared resources are unavailable for pixel processing, then the Selzer/Fiske combination would allocate its thread contexts for vertex processing." EX. 1003, ¶242. However, the disclosures in Selzer and Fiske do not support that.

371. In Selzer, the rendering module (the alleged unified shader) will only begin to process geometry operations if it has no rendering operations to perform. EX. 1009, 47. That is, the only consideration of when to switch is how much work of a given type there is to do. There is no disclosure in Selzer that it switches from rendering operations to geometry operations based on enough available space in any of Dr. Pfister's identified memories. *See id.* Fiske likewise contains no disclosures

that available space is checked prior to executing certain threads; what thread it executes is purely based on priority. EX. 2010, 211.

372. Further, the claims have specific differences beyond the evaluation of storage space that Selzer and Fiske fail to disclose. Claim 1 requires performing vertex operations until there is no more room in the general purpose register block for more incoming vertex data, in which case continuing to perform pixel operations until enough space appears to once again store vertex data. But Selzer's rendering module, the alleged unified shader, does not do that. Instead, it operates on rendering calculations, which Dr. Pfister alleges to be pixel data (EX. 1003, ¶229), and then when it runs out of the alleged pixel data, it operates on geometry calculations, which Dr. Pfister alleges to be vertex data (*id.*). Selzer has no disclosure at all when to cease sending, storing, or operating on the alleged vertex data, much less any description of doing so based on not having room for vertex data in the general purpose register block, nor any description of continuing to execute pixel processing in that scenario. *See* EX. 1009, 47. There is also no description of determining a second time at which there is again room to store vertex data. *See id.*

373. Claim 3 requires the processor unit to be "operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations." Similar to the above, the triggering event in Selzer in going from the alleged pixel operations (rendering

calculations) to the alleged vertex operations (geometry calculations), has nothing to do with any determination that, at a given time, there are finally enough shared resources for geometry calculations; the only consideration is that there are no more rendering calculations to operate on. *See* EX. 1009, 47. There may have been sufficient resources before that happens, but the system has no way of knowing, and thus no way of "perform[ing] pixel calculation operations *until* enough shared resources become available and then us[ing] the shared resources to perform vertex calculation operations," as required by claim 3.

374. Claim 4 is similar to claim 3, but requires the processor unit to be "operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations." There is no disclosure in Selzer of processing the alleged vertex data (geometry calculations) until there is enough room for processing the alleged pixel data (rendering calculations). *See* EX. 1009, 47. There is not even disclosure of a switch in that direction. *Id.* The only switch described for the rendering module is switching from rendering calculations to geometry calculations, not from geometry calculations to rendering calculations. *Id.* And again, even the switch in the wrong direction is only done based on whether or not there is any rendering work to do. *Id.*

2. No Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11)

375. This claim requires the processor unit to switch between unfinished vertex operations and pixel operations. Specifically, claim 11 reads:

A unified shader comprising:

> a processor unit flexibly controlled to perform vertex manipulation operations and pixel manipulation operations based on vertex or pixel workload; and
>
> an instruction store and wherein the processor unit of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store.

EX. 1001, claim 11. Dr. Pfister relies on purely on Fiske, which does not disclose this functionality. EX. 1003, ¶256.

376. Dr. Pfister relies on Fiske's thread priority system. *Id*., ¶238 ("Because Fiske assigns a priority to each thread in the system, the scheduling and instruction selection may be very fine-grained and lead to an efficient and dynamic system that can move back and forth between vertex and pixel operations very rapidly."), ¶¶241-242 (describing Fiske's scheduling algorithm). Fiske explains that "the goal is to identify as exactly as possible a relative order in which threads should be run, and devote as many resources as possible to the most important threads." EX. 1010, 211. Fiske describes loading the highest priority threads to the processor. *Id.* Loaded threads are then scheduled by Fiske's scheduler for execution by the processor. *Id.* This disclosure does not involve switching instructions such that the vertex and pixel operations are at various degrees of compeltion. Dr. Pfister has identified nothing in

Fiske that clearly discloses switching between incomplete threads, much less incomplete individual instructions. His opinions are entirely conclusory, and not tied to any specific disclosure in Fiske. For example, Dr. Pfister claims "Fiske performs another selection to determine which resources should be used for vertex or pixel operations," when in fact Fiske does not even mention vertex or pixel operations. EX. 1003, ¶242*; see generally* EX. 1010.

377. However, to the extent Fiske does disclose switching execution between unfinished threads, and further unfinished instructions, it does not do so between *vertex and pixel* threads and instructions because it is indifferent to vertex and pixel operations; any purported switching occurs only between loaded high priority threads. EX. 1010, 211. Fiske does not discuss graphics processing at all. *See generally id*. Nor does Fiske disclose floating-point or fixed-point operations. *See generally id.*

378. Moreover, there is no motivation to add any of Fiske's functionality to Selzer. Selzer wants its rendering module to render if there is any rendering work to do, because that results in "peak performance." *See* EX. 1009, 47-48. That is, only if it is in an "idle state," which occurs "if there is no rendering data in his input buffer," does the rendering module "request new unprocessed geometry objects and continue geometry calculations." *Id.* For the rendering module to switch between rendering and geometry, it "consumes time." *Id.*, 48. Therefore, a POSITA would

not add the ability to switch between unfinished pixel and vertex threads to Selzer, which would decrease Selzer's efficiency at processing graphics in the manner it deems most efficient and with highest performance, with no upside. *See also* Section XIII.C.7 (explaining no motivation to combine Selzer and Fiske and no reasonable expectation of success in doing so), *infra*.

379. Separately, Dr. Pfister has not explained how Fiske (or Selzer) would switch between incomplete pixel and vertex operations "*based on* switching between instructions in the instruction store." EX. 1001, Claim 11 (emphasis added). Dr. Pfister appears to assert that switching between instructions meets the limitations of claim 11 but does not describe how those instructions are located within an instruction store, which Dr. Pfister alleges is Selzer's master module. *See* EX. 1003, ¶242.

3. No Execution of Instructions "In Response to" Receiving Selected Data (Claim 2)

380. This claim requires executing instructions in response to receiving selected data. Specifically, claim 2 reads:

A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit;

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel

> calculation operations on selected data maintained in the general purpose register block; and
>
> wherein the processor unit executes instructions that generate a pixel color in response to selected data from the general purpose register block and generates vertex position and appearance data in response to selected data from the general purpose register block.

EX. 1001, Claim 2. Selzer's alleged processor unit does not execute instructions in response to selected data.

381. Additionally, Fiske itself states the opposite. The ground relies on Fiske's "thread prioritization scheduling scheme," which is "a simple means of guiding the scheduling of threads on a node," to explain how the alleged combination decides what the processor will execute and when. EX. 1003, ¶230; EX. 1010, 211. Fiske confirms that it is the receipt and loading of threads, or groups of instructions, that cause execution—not receipt of data. "The software scheduler uses the priority to decide *which threads are loaded* . . . the hardware scheduler uses the priority to determine *which loaded thread to execute*." EX. 1010, 211.

382. Further, performing rendering or geometry processing on received the data, without anything more, does not meet the claim requirement. The fact that somehow operations are done on data does not mean that that the "processor unit executes instructions . . . *in response to* selected data." All Selzer discloses that the rendering module receives data, and at some point in time it processes that data. EX. 1009, 41. If the triggering event is the receipt of instructions, as it is in the

combination of Selzer and Fiske, rather the receipt of data, the execution simply is not "in response to selected data," it is simply *on* selected data, in response to instructions.

4. Alleged "Sequencer," "Instruction Store," "Circuitry," "Selection Circuit," and "Arbiter" Are Not Within Alleged "Unified Shader" As Required (Claims 2, 5-7, 10, 11)

383. Claims 2, 5-11 each require a "sequencer" that "maintain[s] instructions" or "instruction store" that stores in instructions, each of which being part of the unified shader. *See* Section XIII.B.4 (providing preamble is limiting), *supra*.

384. The alleged sequencer and the alleged instruction store in Ground 3 are not within the alleged unified shader, and thus the ground fails as to claims 2, 5, and 11. Dr. Pfister identifies Selzer's "rendering module" as the claimed unified shader. EX. 1003, ¶¶227, 232. Dr. Pfister identifies Selzer's master module as the claimed sequencer. EX. 1003, ¶¶236 (in section regarding "sequencer" of claim 2, describing only abilities of master module).

385. But the master module is outside the rendering module, not inside it as required. Selzer describes the master module and the rendering module as "different units." EX 1009, 40. This is also seen in Figures 4.2 and 4.4 of Selzer:

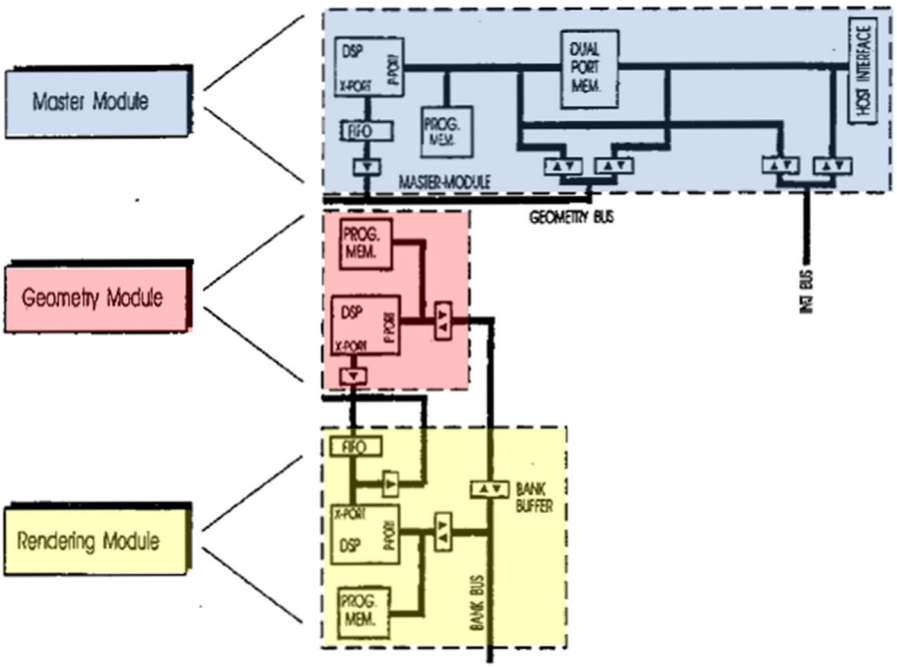
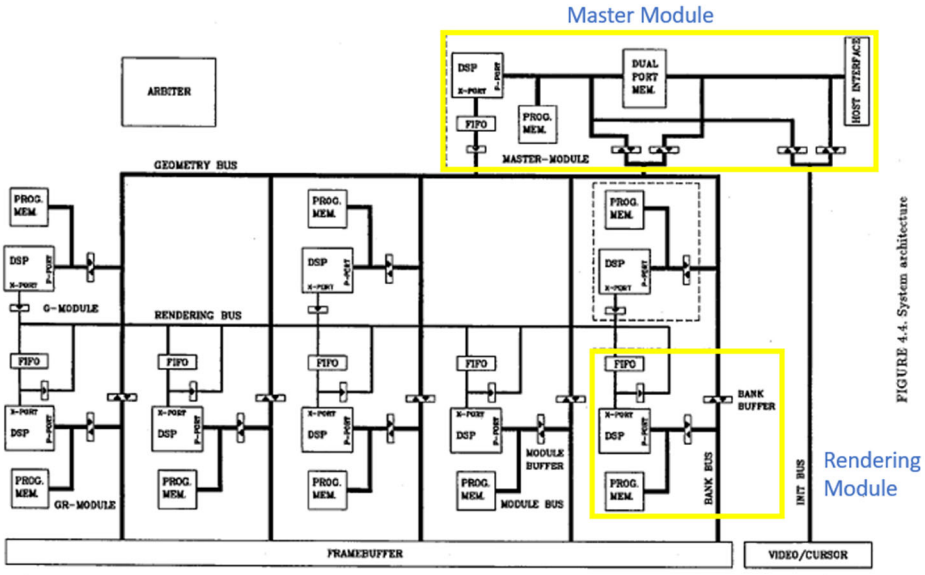


FIGURE 4.2. Basic module structure

EX. 1009, 42 (annotated).



EX. 1009, 44 (annotated).

386. Dr. Pfister identifies Selzer's master module as also meeting the

"circuitry" of claim 6. EX. 1003, ¶249. Claim 6 requires "the sequencer further includes circuitry operative to fetch data from a memory." EX. 1001, claim 6. Thus the alleged "circuitry" of claim 6 within the sequencer of claim 5 is not within the alleged unified shader, either.

387. Dr. Pfister's theory with respect to "selection circuit" and "arbiter." Claim 7 requires "shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal." EX. 1001, claim 7. Meanwhile, claim 10 provides "shader of claim 7, wherein the control signal is provided by an arbiter." *Id.*, claim 10. Dr. Pfister does not clearly identify what it alleges to be the "selection circuit." EX. 1003, ¶252. To the extent that it is the master module, it is also not within the claimed unified shader as required in claim 7 and 10. To the extent Dr. Pfister alleges the arbiter is a component of the master module, *e.g.*, the DSP, it also fails for this reason.

5. Alleged "Sequencer" And "Instruction Store" Do Not Maintain Instructions (Claims 2, 5, and 11)

388. As explained in the section above, Dr. Pfister and its expert allege that the claimed sequencer and instruction store of claims 2, 5, and 11 is met by the "master module" of Selzer. But, separately from the failures described in the section above, the master module of Selzer does not satisfy the requirement of claims 2, 5, and 11 of "maintaining instructions" or storing "instructions in the instruction store." EX. 1001, claims 2, 5, and 11.

389. Dr. Pfister's theory is that "the master module in Selzer accesses the graphics data in the system and 'fetches the next primitive or task automatically.'" EX. 1003, ¶236. This is incorrect. It is the geometry module—not the master module—that performs the function that Dr. Pfister relies on as its evidence of why this limitation is disclosed. EX. 1009, 43 ("If a geometry module has finished the last task, it accesses the geometry bus and fetches the next primitiv[e] or task automatically."). The geometry module wholly distinct from and not a part of the master module, as described above, and shown in Figure 4.2 of Selzer. Thus, the advanced theory fails.

390. Dr. Pfister argues that Selzer also says the master module is the "systems supervisor," that "is responsible for start-up and synchronizing activities." EX. 1003, ¶236. Nothing in that disclosure states that the master module "maintains instructions," as required. EX. 1009, 40. The master module's role is "put[ing] the high order primitives like splines, polygons, meshes or triangles into the *data FIFO*." *Id.* (emphasis added).

391. Dr. Pfister further argues that "[t]he master module gathers the graphics data and tasks and supplies those tasks (data and instructions) to the appropriate processor unit." EX. 1003, ¶236. That characterization is incorrect. All Selzer says is that the master module puts data into the data FIFO. EX. 1009, 41 (defining "The Master Modul [*sic*]" by stating: "The master module traverses the graphics data

structure and feeds graphics data to a special first-in-first-out memory (FIFO) for delivering to the appropriate processors."); 43 ("The master module traverses the graphics data structure and puts the high order primitives like splines, polygons, meshes or triangles into the data FIFO."). As explained above, it is the geometry module that handles instructions.

392. Dr. Pfister further argues that Fiske in combination with Selzer also discloses the sequencer. EX. 1003, ¶¶237-238. This theory builds on Selzer according to Dr. Pfister. *Id.* But Dr. Pfister does not actually identify any component in Fiske as the claimed sequencer, and focuses on adding Fiske to Selzer's rendering module, not the alleged sequencer, the master module. *Id.* The alleged ability of Fiske to assign priorities on a per thread basis does not disclose a sequencer, nor does Dr. Pfister say so. *Id.*

6. No "Arbiter" (Claim 10) or "Control Signal" (Claims 7, 10)

393. Dr. Pfister seemingly identifies the DSP of the master module as meeting the claimed "arbiter" required in claim 10. EX. 1003, ¶251. However, Selzer does not disclose any arbitration abilities by the DSP, nor the provision of a control signal. EX. 1009, 39-40. Thus, Dr. Pfister's theory is unsupported by Selzer's disclosures.

394. Additionally, Dr. Pfister failed to identify any "control signal" outside of conclusory allegations. He opines that the DSP of the master module generates a

control signal but cites to nothing from Selzer. EX. 1003, ¶251. Dr. Pfister cites to Selzer pages 43-45 but these pages do not disclose any control signal that is allegedly generated by the DSP of the master module, nor do any others. EX. 1003, ¶251. Selzer contains only two extremely thin references to the DSP, and neither of them describe anything regarding a control signal. EX. 1009, 39-40.

395. Dr. Pfister also alleges that the combination of Selzer and Fiske discloses claims 7 and 10. EX. 1003, ¶252. Dr. Pfister identifies the master module of Selzer and the scheduling arbiter of Fiske as the selection circuit, but the same issues described above apply here in that Dr. Pfister has not shown that the master module is part of the alleged unified shader, the rendering module.

396. With respect to claim 10, Dr. Pfister identifies Fiske's hardware scheduler as the claimed arbiter and claims that its "simple hardware" provides a control signal. EX. 1003, ¶252. But there is no disclosure of a control signal provided by the "simple hardware" in Fiske. All Fiske states is: "On a context switch, the hardware scheduler chooses the loaded thread with the highest priority as the next thread using simple hardware, in order to minimize context switch overhead." EX. 1010, 211. Nor has Dr. Pfister shown or alleged that the alleged arbitration disclosed by Fiske requires providing a control signal.

7. No Motivation to Combine Selzer and Fiske and no Reasonable Expectation of Success in Doing So

397. Dr. Pfister opines that it would be obvious for a POSITA to combine

Selzer and Fiske because both are directed to dynamic load balancing schemes. EX. 1003, ¶222. But Fiske literally says that it is not: "We do not consider dynamic load balancing issues in this paper." EX. 1010, 211. Thus, a POSITA would not have been motivated to combine the two references.

398. A POSITA would not be motivated to combine Fiske with Selzer because it would be antithetical to Selzer's disclosure that its "peak performance" is when "all rendering processor power is exploited for rendering calculations and the rendering are working continuously." EX. 1009, 48. To a POSITA, it would make no sense to switch away from processing rendering calculations if such calculations were unfinished. Doing so would lose the system's peak performance.

399. The only time it makes sense for Selzer's rendering module to work on geometry is when "there is no rendering data in [the rendering processor]'s input buffer." *Id.*, 47. But Selzer already discloses an adequate task switching ability which maintains its peak performance. *Id.* Thus, adding Fiske solely for the purpose of switching threads, as opposed to tasks, is unnecessary and not something a POSITA would do.

400. There is no reasonable expectation of success in combining Fiske with Selzer. Fiske is directed to a very different type of computing that Selzer. Fiske is focused on highly parallel computing systems where threads interact with each other through communications and synchronization methods. For example, the abstract says

"Multiple-context processors provide register resources that allow rapid context switching between several threads as a means of tolerating long communication and synchronization latencies." EX. 1010, 210. Furthermore, the very first sentence of Fiske's main body says: "Parallel processor performance is critically tied to the mechanisms provided for tolerating long latencies that occur during remote memory accesses, and processor synchronization operations." *Id.* This characterization is consistent with the benchmark software that Fiske used to evaluate the performance of their system including a "number of simple benchmarks in which synchronization performance is critical." *Id.* Fiske says that "[t]hese benchmarks evaluate the performance of barrier synchronization, queue locks, and fine-grain synchronization." *Id.*, 211. I agree with their characterization of these benchmarks and that they represent a particular form of computation that is fundamentally different from geometry processing and vertex rendering.

401. Fiske then goes on to characterize essential program characteristics that represent the key performance focus of their work. "If a critical thread is not loaded, then no progress can be made along the critical path, and runtime performance suffers. If the critical threads are loaded along with other non-critical threads and the scheduler treats all the loaded threads as equal, then runtime performance suffers." *Id.* The notion of critical threads is unique to computing tasks that have real-time performance and synchronization needs.

402. In Section 2.2, Fiske says "Prioritizing threads incorrectly can lead to a

number of deadlock situations. Specifically, if thread A is waiting for another thread B to complete some operation, and thread B has a low priority that does not allow it to be loaded, then deadlock results." *Id.* Deadlock is a known situation where the software on a processor gets into a state where no forward progress can be made, and the entire system is locked in an inoperable state.

403. Additionally, Fiske is based on a processor from the Alewife processor that is derived from a SPARC processor (section 3.2 System Parameters). The SPARC processor was designed by Sun Microsystems and has two different and separate register files, one for storing floating-point information and the other for Integer and logical data. There is no general-purpose register file that stores both types of data.

404. Selzer is fundamentally different from Fiske. Unlike Fiske, Selzer has no concept of thread communication and synchronization. Selzer does not discuss thread priority. Selzer never discusses deadlock issues, which is one of the main concerns of Fiske.

405. There is no evidence in Selzer that what Dr. Pfister identifies as Geometry, *i.e*. floating-point, and Vertex, *i.e.*, fixed-point, share a common register file. Selzer is based upon an unidentified disital signal processor (DSP). At the time of Selzer's publication most DSPs had separate register files for floating-point and integer data.

406. Moreover, Fiske is an underspecified research paper prototype, it was never finalized into hardware and only simulated at a very high-level of abstraction. The multi-context architecture that Fiske models and simulates has an

extremely complex set of implementation details. Not only is it very different from Selzer's unidentified DSP but there are innumerable design details that need to be sorted out requiring a significant amount of engineering design, research, and validation. There is no clear reasonable expectation of achieving success constructing a hybrid architecture that combines Selzer with Fiske.

407. For the above reasons, a POSITA would not have been motivated to combine Selzer and Fiske, nor have a reasonable expectation of success in doing so.

XIV. SECONDARY CONSIDERATIONS SUPPORT NON-OBVIOUSNESS OF THE '454 PATENT

408. In my opinion, the following objective indicia of nonobviousness are relevant to the validity of the '454 patent: (1) initial skepticism, (2) unexpected results, (3) satisfaction of a long-felt need and failed attempts by others, (4) industry respect and praise, (5) commercial success, and (6) adoption by others.

409. The '454 patent is entitled to a presumption of nexus for objective indicia of non-obviousness. Beginning in 2001, ATI developed the unified shader technology of the '454 patent, which included the R400 project, its seamless transition to the R500 project and to the Xenos project, which would ultimately power the highly successful Xbox 360 gaming console (released in 2005). *See, e.g.*, EX. 2127, 99 (David Luebke (of NVIDIA Research) and Greg Humphreys, "How GPUs Work," How Things Work, (Feb. 1, 2007)) ("Unified shaders were first realized in the ATI Xenos chip for the Xbox 360 game console."); EX. 2102

(Lefebvre Depo. Tr.) (June 28, 2017), 132:1-149:25, 198:1-199:25 (testifying that the transition from R400 to Xenos "was very fluid" and that "the Xenos program, as far as the sequencer and the unified shader is concerned, is the R400 program, and such . . . it was just one big continuous [project]"), 240:1-241:25 (testifying that the Xenos sequencer had "a very, very similar arbitration scheme" to the R400 and that the images in the specification describing the various operation points of the Xenos sequencer and the R400 "are identical"), 246:6-14; EX. 2113 (Gruber Depo. Tr.) (July 27, 2017), 210:17-211:10; EX. 2133, 5 ("Working of Xbox 360 – How Xbox 360 Works," (Feb. 11, 2010)) ("[T]he new ATI graphics card in the Xbox 360 has unified shader architecture. What that means is that now, both shader types share the same pipelines."); EX. 2145, 13 (Andrews & Baker, "Xbox System Architecture, IEEE Computer Society, IEEE Micro (March-April 2006)) (referring to the Xbox 360 as "the first game console with GPU-unified shader architecture"); *see generally* EX. 2144 (Dean Takahashi, "*The Xbox 360 Uncloaked: the Real Story Behind Microsoft's Next-Generation Video Game Console*" (SpiderWorks, LLC 2006)).

410. Even if the '454 patent is not entitled to a nexus based on being coextensive with these products, the objective indicia of non-obviousness provided below still are the direct result of the unique characteristics of the inventions claimed in the '454 patent. The asserted objective evidence is tied to a specific product, including but not limited to the following AMD product which is the first

commercial implementation of the unified shader technology claimed by the '454 patent: the Xenos GPU. *See* EX. 2145, 13. The unified shader technology described in the '454 patent is coextensive with the Xenos GPU, such that evidence of the existence of secondary considerations of non-obviousness related to this GPU apply equally to the '454 patent. Even if the '454 patent is not entitled to a nexus based on being coextensive with the Xenos GPU, as further seen below, the objective indicia of non-obviousness provided below still are the direct result of the unique characteristics of the invention claimed in the '454 patent.

A. Initial Skepticism of ATI's Unified Shader Technology

411. The unified shader was greeted with some skepticism by members of the electronic engineering and gaming industries. For example, industry analysts in 2006 were cited as saying that some observers thought that unified shaders were "***not the obvious right answer right now***," and that Microsoft was "taking a chance" on undertaking the "experiment" of unified shaders. *See, e.g.*, EX. 2144, 5 ("Joe Peddie, a graphics expert and analyst at Jon Peddie Research, believed that the unified shaders would be ***an experiment*** for the entire industry. Microsoft would be ***taking a chance*** by being the first to undertake it. The question was whether the unified shaders could balance the workload or not. The system had to efficiently allocate the shaders to either type of processing at any given moment. The unified shaders were less likely to be wasted. But they were complicated. 'Are they more

efficient? The answer is ***yes, no, maybe***,' Peddie said. '***Some observers think unified shaders are not the obvious right answer right now***, that we as an industry just have to learn more about how exploit them, the Xbox 360 is a great laboratory.[']" (emphasis added)).

412. Notably, other companies such as NVIDIA, publicly criticized the idea of a unified shader, minimizing the concept as an "implementation detail" rather than a "feature," and characterizing the concept as a "Jack of all trades, master of none." *See, e.g.*, EX. 2129, 1 (Anton Shilov, "ATI and NVIDIA Proclaim Different Graphics Processors Architecture Goals," X-bit Labs Hardware News, (Dec. 23, 2004)) ("NVIDIA Corp.'s chief architect David Kirk characterized the unified graphics engines as ***an implementation detail, not a feature***, but admitted the unified architecture would be nice for programmers, who would have one instruction set for vertex and pixel shaders. 'It's not clear to me that an architecture for a good, efficient, and fast vertex shader is the same as the architecture for a good and fast pixel shader. A pixel shader would need far, far more texture math performance and read bandwidth than an optimized vertex shader. So, if you used that pixel shader to do vertex shading, most of the hardware would be idle, most of the time. Which is better – a lean and mean optimized vertex shader and a lean and mean optimized pixel shader or two less-efficient hybrid shaders? There is an old saying: '***Jack of all trades, master of none***',' Dr. Kirk said in an interview with ExtremeTech web-

site."); EX. 2130, 1 (Anton Shilov, "NVIDIA Chief Architect: Unified Pixel and Vertex Pipelines – The Way to Go. NVIDIA Says It Would Make a Chip with Unified Pipes 'When it Makes Sense'", X-bit Labs Hardware News, (July 11, 2005)) ("[NVIDIA] will do a unified shader architecture in hardware when it makes sense. When it's possible to make the hardware work faster unified, then of course we will. . . . [B]ut in the meantime, there's plenty of mileage left in [the dedicated shader] architecture. . . . [U]nified shader processor [i]s not really efficient for both vertex and pixel processing compared to specially optimized pipes . . . . [A]t this point NVIDIA thinks the approach is not something required tremendously in short term future, which may be a feasible, as [Windows] Longhorn is not expected to be launched earlier than in mid-2006. 'It's far harder to design a unified processor - it has to do, by design, twice as much. Another word for 'unified' is 'shared', and another word for 'shared' is 'competing'. It's a challenge to create a chip that does load balancing and performance prediction. It's extremely important, especially in a console architecture, for the performance to be predicable. With all that balancing, it's difficult to make the performance predictable,' Dr. Kirk said.").

413. Also, companies like Sony criticized the unified shader developed by ATI. *See, e.g.*, EX. 2146, 5 (Hiroshige Goto, Weekly International News, PC Watch, Interview (3) with SCEI (Sony) President Kutaragi, "Why I Worked With NVIDIA on the PS3," (June 13, 2005)) (with certified translation) ("Question: [']Microsoft

used a Unified-Shader architecture for ATI's GPU in the Xbox 360. Is Unified-Shader more advanced in programmability?[‘] Mr. Kutaragi: [‘]ATI's architecture looks good at first glance because the shader is shared and its architecture is identical to the Vertex Shader and the Pixel Shader, ***but I don't think so***. For example, where do you put the results of vertex processing and how do you flow it back to the shader (for pixel processing). ***The whole thing stalls if any part of it gets clogged up.*** The picture that is being colored is different from reality. When it comes down to practical efficiency, I believe ***NVIDIA's approach is superior***.[’]").

414. Some members in the industry highlighted the difficulties facing ATI in trying to achieve success with its unified shader technology. *See, e.g.*, EX. 2136, 7 (Anand Lal Shimpi and Derek Wilson, "Microsoft's Xbox 360, Sony's PS3 - a Hardware Discussion," AnandTech (June 24, 2005), http://www.anandtech.com/print/1719/) ("With the shift to the unified shader architecture, it is ***even more imperative*** to make sure that everything is running at maximum efficiency. We don't have many details on ATI's ability to context switch between vertex and pixel shader programs on hardware, but suffice it to say that ***ATI cannot afford to have any difficulties in managing threads on any level***…Thread management is likely one of the most ***difficult*** things ATI had to work out to make this hardware ***feasible***." (emphases added)).

415. And some in the industry expressed uncertainty as to the extent of

success the unified shader technology would reach in the future. *See, e.g.*, *id.* ("Performance of [the Xenos GPU] hardware is a ***very difficult aspect to assess*** without testing the system. The potential is there for some nice gains over the current high end desktop part, but ***it is very difficult to know*** how easily software engineers will be able to functionally use the hardware before they fully understand it and have programmed for it for a while. Certainly, the learning curve won't be as steep as something like the PlayStation 2 was (DirectX is still the API), but ***knowing what works and what doesn't will take some time***." (emphases added)); EX. 2124, 32 (Dave Baumann, "ATI Xenos: Xbox 360 Graphics Demystified," Beyond3D (written for Consumer Graphics), (June 12, 2005), https://www.beyond3d.com/content/articles/4/1) ("...[W]e suspect that ***it won't be until the second generation of XBOX 360 titles*** before we see developers being able to seriously scratch the surface of understanding the processing capabilities of Xenos and the XBOX 360 as a whole, given that most of the first generation titles will not have been developed on the final hardware.") (emphasis added). Even within ATI, the inventors of the '454 patent had to convince others about the merits of unified shader technology. *See, e.g.*, EX. 2113, 41:6-44:21 (emphases added):

> A. . . . I do remember a specific trip to Orlando, for instance, to present this idea to the engineers there and to convince them that it was a good idea….
> Q. Do you recall any reactions that you received at that presentation?
> A. I think there was openness, but ***some skepticism*** as well as to whether this idea would actually work.

Q. Why is that?
A. There was ***some concern*** over both the performance and whether it would have adequate precision for some of the operations that needed to happen… The concern was that this shader meant that we were changing the way that we did interpolation. So rather than having the setup engine move the vertices—sometimes a triangle can be large, larger than the screen, and to draw it, you only want to draw the portion that is on the screen. So you have to clip the triangle to the screen boundaries. The way that we had previously done that was in the setup engine we would actually generate new vertices at the screen boundary. So that would be done in the setup engine. The unified shader idea was that we would take the original vertices and interpolate them within the shader from their original positions. So that sequence of operations meant that there were ***different precision outputs, different precision results*** you would see, and the issue was ***whether that would cause a problem in terms of accuracy of the interpolation***.

B. Unexpected Results in Developing the Unified Shader Technology

416. ATI's revolutionary efforts in developing its unified shader technology amounted to significant, innovative, and disruptive advances over existing graphics processors. When details about ATI's unified shader became known through the Xenos GPU in the Xbox 360, the "fresh perspective" of ATI "force[d]" those in the industry to "reconsider how we think of the overall pipeline" even though "[c]onceivably" there was "no reason" why the current development model couldn't continue to exist in the PC space. *See, e.g.*, EX. 2124, 1-2 ("[T]he designs of ***contemporary graphics processors still bear the fundamental similarities to their forebears***: vertex processing up one end of the pipeline, pixel processing down the other[,] and still very much aligned with multiples of pixel pipelines. ***Conceivably there is no reason*** why this development model couldn't continue to exist in the PC

space…However, ATI have multiple design teams working on ***different architectures*** concurrently, so whilst their PC processors may follow a fairly familiar lineage[,] other parts of the company have been talking this shader era with ***a completely fresh perspective*** in order to consider the needs of a 'Programmable Graphics Processor'... ***In doing so they will force us to reconsider how we think of the overall pipeline*** and make initial performance assessments based upon 'pipelines' alone.") (emphases added).

417. Also, during the development of the unified shader technology, ATI developers themselves were reported to experience some unexpected results. *See e.g.*, EX. 2144, 5 ("In the ATI design, the unified shader was smart enough to juggle between the two types of shaders. The graphics chip would have 48 unified shader pipelines running at the same time[, which]…was better than the 32 [unified shader pipelines] the team had expected."); EX. 2147, 1 (Scott Wasson, "Details of ATI's Xbox 360 GPU unveiled," The Tech Report, (May 19, 2005), http://techreport.com/review/8342/details-of-ati-xbox-360-gpu-unveiled) ("The GPU can dynamically allocate shader resources as necessary in order to best address a computational constraint, whether that constraint is vertex- or pixel-related. This sort of graphics architecture has been rumored as a future possibility for some time, but ***ATI worried that using unified shaders might cause some efficiency loss***. To keep all of the shader units utilized as fully as possible, the design team created a

complex system of hardware threading inside the chip itself.") (emphases added).

C. The Satisfaction of a Long-Felt Need by the Unified Shader of the Xenos GPU and Failed Attempts by Others

418. The Xbox 360, with its ATI Xenos GPU, was a fundamentally different approach than the current graphics technology available at the time of its launch. *See*, *e.g.*, EX. 2124, 3 (noting that the Xenos graphics processor is "***different from others seen before in the PC space***," is "***very different from even ATI's impending new PC graphics components***," "***doesn't directly correspond to any current graphics processor***[,]" and has "***a fundamental difference with virtually all current shader capable graphics processors***…***until now*** [shader processing has] done both [geometry and raster pipeline] elements distinctly") (emphases added); EX. 2145, 8 ("***Unlike earlier graphics engines***, the shaders are dynamically allocated, meaning that there are no distinct vertex or pixel shader engines—the hardware automatically adjusts to the load on a fine-grained basis."); EX. 2147, 1 ("[Bob] Feldstein [the Vice President of Engineering at ATI who oversaw the Xbox 360 GPU project] cited several ***major areas of innovation where the Xbox 360 GPU breaks new ground*** [including]…the unified shader array…In ***traditional*** pixel shaders, he noted, any shader output is generally treated as a pixel, and it's fed through the rest of the graphics pipeline after being operated on by the shader. ***By contrast***, the Xbox 360 GPU can take data output by the shaders, unaltered by the rest of the graphics pipeline, and reprocess it. This ***more efficient*** flow of data, combined with a unified

instruction set for vertex and pixel manipulation, allows ***easier implementation*** of some important graphics algorithms in real time, including higher-order surfaces and global illumination.") (emphases added).

419. This unified shader approach of the Xenos GPU satisfied long-felt needs in the industry "[f]or years," where others failed to satisfy those needs. *See, e.g.*, EX. 2136, 6 ("***At this point in time***, much of the bandwidth generated by graphics hardware is required to handle color and z data moving to the framebuffer. ATI hopes to ***eliminate this as a bottleneck*** by moving this processing and the back framebuffer off the main memory bus…***For years***, vertex and pixel shader hardware have been implemented separately, but ATI has sought to combine their functionality in a unified shader architecture…The GPU in the Xbox 360 uses ***a different architecture than we are used to seeing***.") (emphases added); EX. 2133, 5 ("***Traditionally***, pixel shaders and vertex shaders have dedicated pipelines because each one has very specific and differing needs. As we learned before, ***the new ATI graphics card*** in the Xbox 360 has unified shader architecture. What that means is that ***now***, both shader types share the same pipelines. ***ATI figured out a way*** at the hardware level to address the needs of both types of shaders using the same pipeline. The apparent advantage of sharing pipelines is to add more assembly lines, making computation that much faster. ATI claims that this unified shader architecture allows for 48 billion shader operations per second. The Xbox 360 is the ***first device***

to use this type of architecture.") (emphases added).

D. Industry Praise of the Unified Shader in Xbox 360's Xenos GPU

420. Despite some initial skepticism, ATI's unified shader technology contained in the Xenos graphics infrastructure of the Xbox 360 evetually received praise in the industry, including through press reports and industry analyst reports, including around the time of the Xbox 360's release in 2005. *See e.g.*, EX. 2131, 3 (Peter Lewis, "Playing for Keeps," CNN Money (for Fortune) (Nov. 17, 2005), http://money.cnn.com/2005/11/17/technology/xbox_fortune_112805/) ("The ***enhanced realism*** comes from the Xbox 360's custom-designed IBM PowerPC processor and ATI graphics chip, which together deliver ***near-cinematic quality***.") (emphases added); EX. 2134, 8 (Ben Kuchera, "Xbox 360: the Ars Technica Review," Ars Technica (Nov. 30, 2005), https://arstechnica.com/gadgets/2005/11/xbox360/8/) ("…***[T]he graphics are almost photo realistic***…There's a lot to like about the system, [including] the ***generational leap in graphical quality***…***The bar has been raised in terms of console graphics***[.]") (emphases added); EX. 2136, 7 ("Regardless of the paths that lead to this road, it is obvious that the Xbox 360 will be ***a geometry power house***.") (emphasis added); EX. 2133, 4 ("Another ***powerful asset*** in the Xbox 360 is the Graphics Processor Unit (GPU)…the ***most innovative*** thing about this card is that it is built on unified shader architecture.") (emphases added); EX. 2140, 1-3 ("Xbox

360 GPU," NeutralX2 (Apr. 26, 2006), https://neutralx2.com/2006/04/26/xbox-360-gpu) ("[T]hanks to its ***innovative use of Unified Shader Architecture*** . . . ATI and Microsoft have truly built ***something special*** in the Xbox360's GPU.").

421. Some of the praise for ATI's unified shader even came from competitior NVIDIA. *See, e.g.*, EX. 2127, 99 ("The net result is a flexible parallel architecture that improves GPU utilization and provides ***much greater flexibility*** for game designers.") (emphasis added).

422. After the release of the Playstation 3 ("PS3"), often cited as the Xbox 360's main rival in the gaming console arena, several articles compared the graphics infrastructures of the two consoles, and declared Xbox 360's Xenos graphics card as superior. *See*, *e.g.*, EX. 2138, 5 (Jesse Schedeen, "Xbox 360 vs. PlayStation 3: The Hardware Throwdown," IGN, (Aug. 26, 2010), http://www.ign.com/articIes/2010/08/26/xbox-360-vs-playstation-3-the-hardware-throwdown) (comparing the ATI Xenos graphics card of the Xbox 360 with the NVidia RSX graphics card of the PS3, and declaring the Xbox 360 as the winner in "The GPU" category, "based on the ***peak performance*** of the eDRAM and the general versatility of this setup") (emphasis added); EX. 2139, 1 ("Xbox 360 and Playstation 3's Graphics Card (GPU) Compared," RedGamingTech, (Mar. 11, 2013), http://www.redgamingtech.com/xbox-360-and-playstation-3s-graphics-card-gpu-compared/) ("The Xbox 360's GPU ***packs a little more punch*** than Sony's

PS3…[T]he Microsoft console came out slightly ahead…The Graphics card on the XBox 360 is based on an AMD design, and the system has 48 unified shaders, running at 500MHZ. This gives 96 billion shader operations per second on the XBox 360, compared to that of the 74.8 billion shader operations that the PS3 managers, with its 24 pixel shader pipelines and 8 vertex shader pipelines running at 550MHZ per second. In terms of raw performance, ***the PS3 loses slightly right out of the gate***.") (emphases added).

E. Commercial Success of the Xbox 360 Containing ATI's Unified Shader in the Xenos GPU

423. Despite some initial skepticism and uncertainty, the new unified shader technology in the Xenos GPU of the Xbox 360 experienced significant commercial success, with the Xenos GPU and the Xbox 360 generating billions of dollars in revenue for ATI and Microsoft. *See e.g.*, EX. 2132, 13 (Graham Singer, "History of the Modern Graphics Processor, Part 3," TechSpot (Apr. 10, 2013)) ("ATI's revenue rose to a record $2.2 billion for the year, the highest in the company's history, aided by shipments of Xenos GPUs for the Xbox 360."); EX. 2128, 4 (Microsoft Corporation FY 2006 Annual Report (Aug. 22, 2006)) ("With 5 million units sold to date, Xbox 360 is ***the fastest-selling video game console ever***. It helped drive revenue in the Home and Entertainment business to $4.26 billion, from $3.14 billion in 2005.") (emphases added); EX. 2135, 1 (Nathan Pensky, "Microsoft Xbox dominates a sluggish 2011 gaming market," Vator, (Jan. 13, 2012),

http://vator.tv/news/2012-01-13-microsoft-xbox-dominates-a-sluggish-2011-gaming-market) (noting that in 2011, Microsoft sold $***2.1 billion*** in consoles).

424. In fact, the Xbox 360 accounted for ***65%*** of the ***global market share*** of video game consoles in 2006, followed by 22% in 2007, 23% in 2008, 23% in 2009, and 28% in 2010. *See, e.g.*, EX. 2142, 1 ("Global Market Share of Game Consoles from 2006-2010," Statista, https://www.statista.com/statistics/272048/global-market-shares-of-game-consoles/); EX. 2135, 1 (noting that in December 2011, for 10 straight months "Xbox 360 held over 40% of the current-generation console market share," and that for the 12 straight months, Xbox 360 was "***the best-selling console***" with Xbox 360 holding 46% of the current-generation console market share) (emphasis added).

425. The commercial success of the Xbox 360 is also attributable to the unified shader technology of the '454 patent in its Xenos GPU. This is evidenced, for example, by the importance of the GPU with respect to the viability of the Xbox 360 in the market and consumer preferences. *See e.g.*, EX. 2136, 10-11 ("…Microsoft is quite concerned with keeping costs down to a minimum this time around… In the end it seems that Microsoft was more focused on spending money where it counts (*e.g.* CPU, ***GPU***, HDD) and skimped on areas that would have otherwise completed the package (*e.g.* more USB ports, built in wireless, router functionality, flash card readers, HDMI support in the box, etc...).") (emphasis

added); EX. 2133, 4 ("Another ***powerful asset*** in the Xbox 360 is the Graphics Processor Unit (GPU)…the ***most innovative*** thing about this card is that it is built on unified shader architecture.") (emphases added).

F. Adoption by Others

426. AMD's unified shader has been widely adopted by others in the industry. Since AMD's innovation in unified shaders, the industry has widely adopted unified shaders in graphics processing technology, including in the GPUs of competitors such as NVIDIA and Qualcomm.

427. Qualcomm adopted ATI's unified shader technology for its Adreno GPU. *See, e.g.*, EX. 2137, 2 (Ryan Shrout, "Qualcomm History and its GPU ®evolution," PC Perspective) ("The mobile GPU market grew rapidly from 2006 into 2009. Qualcomm admits that these were the most difficult years of development, caused by the massive growth of the market. Though the company had talented engineers that were working on GPU technology, the speed of the market shift forced Qualcomm to look for help outside the company, to ATI (now AMD). Together they developed the Adreno 130 — an upgrade from Qualcomm's own in-house designed Adreno 120 GPU. The partnership expanded when Qualcomm licensed a GPU from ATI, called it Adreno 200, and later made some upgrades to create the Adreno 205."). *See* EX. 2141, 2 (Adreno, Wikipedia, https://en.wikipedia.org/wiki/Adreno) (showing Adreno 130 had a "fixed function

pipeline" whereas Adreno 200 and later had a "Unified shader model 5-way VLIW").

428. Moreover, the NVIDIA also adopted a unified shader in its GeForce GPUs. EX. 2143, 2 (NVidia, GeForce 8600/8500 FAQ, (Apr. 29, 2007)) ("Historically, GPUs have had dedicated units for different types of operations in the rendering pipeline, such as vertex processing and pixel shading. With the ***unified architecture of the GeForce 8Series***, NVIDIA designed a single floating poi nt shader core with multiple independent processors. Each of these independent processors can handle any type of shading operation, including pixel shading, vertex shading, geometry shading, and physics shading. GeForce 8 Series GPUs dynamically allocate processing power depending on the workload of the application, providing unprecedented performance and efficiency.").

XV. CONCLUSION

429. In signing this declaration, I recognize that it will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place in the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

430. The statements and opinions in this declaration comprise and are based

upon my review, analysis and understanding of the source materials that I have reviewed as discussed herein, based on the disclosure and statements made within such source materials.

431. I reserve the right to supplement my opinions in the future to respond to any arguments that the Petitioner raises and to consider new information as it becomes available to me.



Dated: February 23, 2024

William Mangione-Smith

APPENDIX A

William Henry Mangione-Smith
4146 118th Ave NE
Kirkland WA, 98033
(425) 654-1424
billms@gmail.com

Highlights

Reviewed and valued thousands of patents.
Priced over $500M worth in patent portfolios.
Key technical support for licensing that has raised more than $1B.
Significant experience developing claim charts and evidence of use.
Former tenured professor at the University of California.
Deposition and trial testifying experience.

Education

BSE (1987), MSE (1992), & Ph.D. University of Michigan
Doctoral Thesis 1992: Performance Bounds and Buffer Space Requirements for Concurrent Processors

Employment

2009-Present	Sole Proprietor of Phase Two LLC – Consulting and intellectual property services All IP related work since 2009 has been conducted under the auspices of Phase Two
2012 May-June	CTO Computers and Consumer Electronics at IP Navigation Group
2005-2008	Intellectual Ventures, Bellevue Washington
2005	Patent valuation, licensing support, focus on computer patents
2006	Head of all IV valuation efforts – managed a team of 12
2007	Director of technology, outbound licensing in consumer electronics
2008	Director of investment strategy, inventor relations
2001-2005	Associate Professor, EE UCLA
1995-2001	Assistant Professor, EE UCLA Focus low power embedded multimedia communications systems
1993-1995	Motorola Wireless Data Group Systems architect for General Magic wireless PDA

1991-1992	Motorola Corporate Research Parallel computer performance monitoring Low power processor architecture
1986-1987	Chrysler Corporation Software Architect

Patents

9,760,588 - Cross-media storage coordination
9,747,426 - Handling masquerading elements
9,716,548 - Data center with free-space optical communications
9,692,887 - Component information and auxiliary information related to information …
9,683,884 - Selective audio/sound aspects
9,680,699 - Evaluation systems and methods for coordinating software agents
9,659,188 - Obfuscating identity of a source entity affiliated with a communique …
9,641,537 - Conditionally releasing a communique determined to be affiliated with a …
9,583,141 - Implementing audio substitution options in media works
9,513,157 - Selective audio/sound aspects
9,479,535 - Transmitting aggregated information arising from appnet information
9,455,035 - Management of memory refresh power consumption
9,426,387 - Image anonymization
9,374,242 - Using evaluations of tentative message content
9,372,536 - Touch screen with tactile feedback
9,367,833 - Data services outsourcing verification
9,306,975 - Transmitting aggregated information arising from appnet information
9,274,582 - Power consumption management
9,230,601 - Media markup system for content alteration in derivative works
9,219,815 - Identifier technique for communication interchange
9,215,512 - Implementation of media content alteration
9,191,341 - Packet routing within an on-chip network
9,178,911 - Evaluation systems and methods for coordinating software agents
9,158,771 - Component information and auxiliary information related to information …
9,152,928 - Context parameters and identifiers for communication
9,092,928 - Implementing group content substitution in media works
9,065,979 - Promotional placement in media works
9,008,117 - Cross-media storage coordination
9,008,116 - Cross-media communication coordination
8,984,579 - Evaluation systems and methods for coordinating software agents
8,984,133 - Providing treatment-indicative feedback dependent on putative …
8,966,630 - Generating and distributing a malware countermeasure
8,949,337 - Generation and establishment of identifiers for communication
8,943,267 - Management of memory refresh power consumption

8,929,208 - Conditionally releasing a communique determined to be affiliated …
8,924,975 - Core selection for applications running on multiprocessor systems …
8,913,753 - Selective audio/sound aspects
8,910,033 - Implementing group content substitution in media works
8,886,577 - Feedback during surgical events
8,850,044 - Obfuscating identity of a source entity affiliated with a communique in …
8,839,255 - Scheduling of threads by batch scheduling
8,819,686 - Scheduling threads on different processor cores based on …
8,818,196 - Data center with free-space optical communications
8,799,912 - Application selection of memory request scheduling
8,774,637 - Data center with free-space optical communications
8,732,087 - Authorization for media content alteration
8,730,836 - Conditionally intercepting data indicating one or more aspects …
8,712,250 - Data center with free-space optical communications
8,682,982 - Preliminary destination-dependent evaluation of message content
8,655,187 - Data center with free-space optical communications
8,640,248 - Handling masquerading elements
8,640,247 - Receiving an indication of a security breach of a protected set of files
8,627,402 - Evaluation systems and methods for coordinating software agents
8,626,848 - Obfuscating identity of a source entity affiliated with a communique …
8,626,731 - Component information and auxiliary information related to …
8,613,095 - Smart distribution of a malware countermeasure
8,607,336 - Evaluation systems and methods for coordinating software agents
8,607,234 - Batch scheduling with thread segregation and per thread type marking …
8,601,530 - Evaluation systems and methods for coordinating software agents
8,601,207 - Management of memory refresh power consumption
8,601,104 - Using network access port linkages for data structure update decisions
8,588,618 - Data center with free-space optical communications
8,583,553 - Conditionally obfuscating one or more secret entities with respect to …
8,559,307 - Routing packets in on-chip networks
8,555,396 - Authenticatable displayed content
8,549,077 - Usage parameters for communication content
8,539,581 - Efficient distribution of a malware countermeasure
8,516,300 - Multi-votage synchronous systems
8,483,569 - Data center with free-space optical communications
8,281,036 - Using network access port linkages for data structure update decisions
8,255,745 - Hardware-error tolerant computing
8,243,045 - Touch-sensitive display device and method
8,224,930 - Signaling partial service configuration changes in appnets
8,224,907 - System and method for transmitting illusory identification characteristics
8,214,191 - Cross-architecture execution optimization
8,209,755 - Signaling a security breach of a protected set of files
8,209,524 - Cross-architecture optimization

8,203,609 - Anonymization pursuant to a broadcasted policy
8,191,140 - Indicating a security breach of a protected set of files
8,161,232 - Periodically and empirically determined memory refresh intervals
8,126,938 - Group content substitution in media works
8,126,190 - Targeted obstrufication of an image
8,117,654 - Implementation of malware countermeasures in a network device
8,082,225 - Using destination-dependent criteria to guide data transmission decisions
8,065,404 - Layering destination-dependent content handling guidance
8,055,797 - Transmitting aggregated information arising from appnet information
8,055,732 - Signaling partial service configuration changes in appnets
8,055,648 - Managing information related to communication
7,972,557 - Plasmon filter
7,930,389 - Adaptive filtering of annotated messages or the like
7,877,584 - Predictive processor resource management
7,860,887 - Cross-media storage coordination
7,752,255 - Configuring software agent security remotely
7,739,524 - Power consumption management
7,653,834 - Power sparing synchronous apparatus
7,607,042 - Adjusting a processor operating parameter based on a performance criterion
7,519,995 - Programmable hardware for deep packet filtering
7,512,842 - Multi-voltage synchronous systems
7,493,516 - Hardware-error tolerant computing
5,793,991 - Method of equalizing loads on a computer bus

Patent Applications

Dr. Mangione-Smith has no known financial interest in any pending US patent applications other than some applications where he is listed as an inventor. The list of such published patent applications where he is a named inventor can be found by searching the online USPTO database.

Expert Consultant in IP Matters – Multi-Year Engagements Noted Once

2023

NETLIST, INC, vs. Micron Technology Inc
Eastern District of Texas, Marshall Division
Civil Action No. 2 o. 2:22-cv-203-JRG
Engaged on behalf of Netlist, counsel is with Irell & Manella LLP

2022

VLSI v. Intel CA 5:17-cv-05671-BLF
Engaged by Irell and Manella LLP on behalf of VLSI

Provided written testimony and was deposed.

Engaged by Mintz Levin on behalf of Lumenis Be Ltd.
Provided written declaration.

DivX, LLC v. Harman International Industries, Inc.,
Engaged by Morrison & Foerster LLP on behalf of DivX, LLC
Provided written testimony and was deposed.

Certain Integrated Circuits, Mobile Devices Containing the Same, and Components Thereof, Inv. No. 337-TA-1335
Engaged by Mintz Levin on behalf of Complainant Daedalus Prime LLC
Provided written testimony, deposed, and at trial.

ITC 337-TA-1318
Advanced Micro Devices, Inc. and ATI Technologies and TCL et. al
Engaged by Mintz Levin on behalf of AMD

NETLIST, INC, vs. SAMSUNG ELECTRONICS CO., LTD.
Eastern District of Texas, Marshall Division
Civil Action No. 2:21-CV-463-JRG
Engaged on behalf of Netlist, counsel is with Irell & Manella LLP
Provided testimony via report, deposition and testimony at trial.

WSOU Investments, LLC d/b/a Brazos Licensing and Development v. ZTE
CIVIL ACTION 6:20-CV-00487-ADA
CIVIL ACTION 6:20-CV-00488-ADA
CIVIL ACTION 6:20-CV-00489-ADA
CIVIL ACTION 6:20-CV-00490-ADA
CIVIL ACTION 6:20-CV-00491-ADA
CIVIL ACTION 6:20-CV-00492-ADA
CIVIL ACTION 6:20-CV-00493-ADA
CIVIL ACTION 6:20-CV-00494-ADA
CIVIL ACTION 6:20-CV-00495-ADA
CIVIL ACTION 6:20-CV-00496-ADA
CIVIL ACTION 6:20-CV-00497-ADA
Engaged by Kasowitz Benson Torres LLP on behalf of WSOU Investments
Provided written testimony and was deposed.

WSOU Investments, LLC d/b/a Brazos Licensing and Development v. Google LLC
CIVIL ACTION 6:20-CV-00571-ADA
CIVIL ACTION 6:20-CV-00572-ADA
CIVIL ACTION 6:20-CV-00573-ADA

CIVIL ACTION 6:20-CV-00575-ADA
CIVIL ACTION 6:20-CV-00576-ADA
CIVIL ACTION 6:20-CV-00579-ADA
CIVIL ACTION 6:20-CV-00580-ADA
CIVIL ACTION 6:20-CV-00583-ADA
CIVIL ACTION 6:20-CV-00584-ADA
CIVIL ACTION 6:20-CV-00585-ADA
Engaged by Folio Law Group on behalf of WSOU
Provided written testimony and was deposed.

Highlands Advisory
IP evaluation

Interdigital
IP evaluation

KONINKLIJKE KPN N.V., v. XIAOMI CORPORATION et al, and COOLPAD TECHNOLOGIES INC.,
The District of Delaware
C.A. No. 21-cv-41-LPS-CJB and C.A. No. 21-cv-43-LPS-CJB
Engaged on behalf of KPN
Counsel is Sussman & Godfrey
Provided declaration.

2021

WSOU Investments, LLC d/b/a Brazos Licensing and Development v. Microsoft Corp., Case No. 6:12-cv-463 (W.D.T.X.)
Engaged by Susman Godfrey on behalf of WSOU Investments
Provided testimony via declaration

KONINKLIJKE KPN N.V., vs.TELEFONAKTIEBOLAGET LM ERICSSON AND ERICSSON INC.
Case 2:21-cv-113 in the Eastern District of Texas
Engaged by Susman Godfrey LLP on behalf of Koninklijke KPN
Testified via declaration and deposition as well as at trial.

Barkan Wireless IP Holdings, L.P. v. T-Mobile US, Inc., T-Mobile USA, Inc., and Nokia of America Corp., No. 2:21-cv-00034-JRG (E.D. Tex.)
Engaged on behalf of Barkan by Heim, Payne & Chorush, LLP
Technology relates to communications equipment.
Provided testimony via declaration and deposition.

Intel Corporation v. FG SRC LLC et al.
Engaged on behalf of FG SRC LLC, by DiMuroGinsberg PC
IPR case number IPR2020-01449
Technology relates to reconfigurable computing
Provided testimony via declaration and deposition.

StratosAudio, Inc. v. multiple automotive manufacturers
Engaged on behalf of StratosAudio, Inc by White & Case LLP
Technology relates to accounting for content distribution

2020

Netlist vs. S. K. Hynix
Nos. 6:20-cv-00194-ADA and 6:20-cv-00525-ADA (W.D. Tex.)
Engaged on behalf of Netlist, provided testimony on behalf of Netlist
Technology relates to memory modules

INTEL CORPORATION, Petitioner v. PACT XPP SCHWEIZ AG
Before the Patent Trial and Appeal Board of the US Patent and Trademark Office
Working with Quinn Emanuel Urquhart & Sullivan on behalf of PACT XPP

Digital Video-Capable Devices and Components Thereof, USITC 337-TA-1224
Working with Mintz, Levin, Cohn, Ferris, Glovsky and Popeo, P.C
On behalf of Koninklijke Philips N.V. and Philips North America LLC
Provided sworn testimony

UNILOC 2017, LLC et al. v. GOOGLE LLC
Northern District of California
4:20-cv-04355-YGR
Prince Lobel Tye LLP
Working on behalf of Uniloc

UNILOC 2017, LLC et al. v. GOOGLE LLC
Working with Ethridge Law Group and Carter Arnett on behalf of Uniloc
Eastern District of Texas, Marshall Division
Case No. 2-18-cv-00492-JRG-RSP, filed 2018-11-17
Case No. 2-18-cv-00496-JRG-RSP, 2018-11-17
Case No. 2-18-cv-00499-JRG-RSP, 2018-11-17
Patent consulting thus far, testimony anticipated
INTEL CORPORATION, v. VLSI TECHNOLOGY LLC
IPR2019-01197, Patent 6,212,633
Working with LOWENSTEIN & WEATHERWAX LLP on behalf of VLSI
Provided expert testimony in the form of a declaration and deposition.

Consulting work for Global Foundries related to computer chips.

GPU++, LLC v. Qualcomm Incorporated, et al.,
Working with Steptoe & Johnson on behalf of GPU++
Texas Western District, case 6:19-cv-00474-ADA, filed 2019-08-16
Testimony provided.
Technology relates to graphics processing systems

VLSI Technology v. Intel
Western District of Texas, case 6:19-CV-256
Working with Irell & Manella LLP on behalf of VLSI Technology LLC
Provided testimony in the form of expert reports and depositions.

2019

VLSI Technology LLC v. Intel
District of Delaware, case 1:18-cv-00966
Working with Irell & Manella LLP on behalf of VLSI Technology LLC
Technology relates to semiconductor manufacturing
Patent consulting thus, testimony anticipated

Consulting work for DivX – related to computer video.

2018

INVT SPE LLC v. Apple Inc., HTC Corp. and HTC America, Inc. ZTE Corp. and ZTE (USA) Inc.
Certain LTE- and 3G-Compliant Cellular Communications Devices, Investigation No. 337-TA-1138
Provided Expert Testimony on behalf of INVT
Technology relates to wireless communications

Mahltig Management- und Beteiligungsgesellschaft mbH in patent proceedings relating to European Patent 1 714 229
Adverse to Intel and Lenovo
Technology relates to microprocessors

Netlist vs. S. K. hynix
US ITC 337-TA-1089
Engaged on behalf of Netlist, provided testimony on behalf of Netlist
Technology relates to memory modules

Voip-Pal.com Inc.
Technology relates to VOIP service providers
Voip-Pal.com, Inc. v. Amazon.com, Inc. et al - 2:18-cv-01076
Voip-Pal.com, Inc. v. Apple, Inc. - 2:16-CV-00260
Voip-Pal.com, Inc. v. Apple, Inc. - 2:18-cv-00953
Voip-Pal.com Inc. v. Verizon and AT&T - 2:16-CV-00271
Voip-Pla.com, Inc. v. Twitter - 2:16-CV-02338

2017

AMD v. LG, Vizio, MediaTek and Sigma Designs
US ITC 337-TA-1044
Provided Expert Testimony on behalf of AMD
Technology relates to graphics processing hardware

Netlist vs. S. K. hynix
US ITC 337-TA-1023
Engaged on behalf of Netlist
Provided Expert Testimony via Report, Deposition and at hearing
Technology relates to memory modules

Advanced Micro Devices Inc.
Engaged on behalf of AMD
USPTO Re-Exam Defense
Provided testimony via expert report and interview
Technology relates to microprocessors

Advanced Micro Devices Inc.
Engaged on behalf of AMD
IPR Defense
Provided testimony via expert report for IPR2015-00321
IPR2015-00328 and IPR2015-00321
Technology relates to microprocessors

2016

INTEL CORPORATION v. FUTURE LINK SYSTEMS, LLC - C.A. No. 14-377-LPS
Retained by Irell and Manella LLP on behalf of Future Link Systems, LLC
Technology relates to computer communications systems
Declarations filed on behalf of Future Link Systems, LLC

Voip-Pal.com Inc.
Technology relates to VOIP service providers
Retained to assist during an IPR action

IPR2016-01198 and IPR2016-01201

TiVo Inc. v. Samsung Electronics Co., Ltd., et al., Case No. 2:15-cv-1503 (E.D. Tex.)
Technology relates to digital video recorders
Retained by Irell and Manella LLP on behalf of TiVo Inc.

Rosetta-Wireless Corporation Inter Partes Reviews Nos. IPR2016-00616 and IPR2016-00622
Technology relates to personal communications and computing devices. Retained by Heim, Payne and Chorush on behalf of Rosetta-Wireless Declaration filed.

Innovative Memory Solutions, Inc. (IMS) v. Micron Technology
District of Delaware - 14-cv-01480-RGA
Engaged by IMS via Cohen & Grace, LLC
Technology relates to semiconductor memory manufacturing

Zak v. Facebook
Eastern District of Michigan - 4:15-cv-13437
Engaged by Zak via Harness, Dickey & Pierce PLC
Technology relates to client-server software Declaration filed.

2015

Advanced Silicon Technologies, LLC Delaware District Court and at the ITC (No matter number assigned)
Technology relates to programmable processor hardware Accused parties:

- Texas Instruments Incorporated 1:15-cv-01175
- NVIDIA Corporation 1:15-cv-01177
- Renesas Electronics Corporation et al. 1:15-cv-01176
- Fujitsu Ten Limited et al. 1:15-cv-01174
- Bayerische Motoren Werke AG et al. 1:15-cv-01178
- VOLKSWAGEN AG 1:15-cv-01181
- Harmon International Industries Incorporated et al. 1:15-cv-01173
- Honda Motor Co., Ltd. et al. 1:15-cv-01179
- Toyota Motor Corporation 1:15-cv-01180

FlatWorld Interactives LLC v. Samsung and LG
C.A. No. 12-804·-LPS and C.A. No. 12-964-LPS
Engaged on behalf of FlatWorld
Technology relates to user interface design

Ericsson Inc. vs. Apple Inc.
EDTX 2:15-cv-290 and 2:15-cv-292
Engaged on behalf of Ericsson
Technology relates to peripheral devices

Synaptics vs. Goodix
US ITC 337-TA-957
Engaged on behalf of Goodix
Provided Expert Testimony via Report, Deposition and at hearing
Technology relates to touch screens

Advanced Micro Devices Inc.
Engaged on behalf of AMD
IPR Defense
Provided testimony via expert report
Technology relates to digital video
IPR2015-00328 and IPR2015-00321

Solocron Media LLC v. Verizon Communications, Inc., Cellco Partnership d/b/a/
Verizon Wireless, AT&T Mobility LLC, T-Mobile USA, Inc., and Sprint Spectrum L.P.
The Eastern District of TX, Case 2:13-cv-1059-JRG
Retained by tensegrity on behalf of Solocron
Technology relates to MMS and multi-media transcoding.
Prepared expert reports on infringement.

Enterprise Systems Technologies S.a.r.l v. Apple Inc., Cirrus Logic Inc., HTC
Corporation, LG Electronics, Inc., Samsung Electronics Co
The International Trade Commission of the United States of America
Inv. No. 337-TA-924
Retained by Mintz, Levin, Cohn, Ferris, Glovsky and Popeo LLP for complainant.
Technology relates to device drivers. Prepared
expert reports and deposed.

Ziilabs Inc., LTD., v. Samsung and Apple
Case 2:14-cv-00203 In the Eastern District of Texas Marshall Division
Retained by Davis of Heim, Payne & Chorush, L.L.P. for Ziilabs Technology
relates to graphics systems.
Provided expert report on infringement as well as deposition testimony.

Packet Intelligence LLC v. Cisco
Case 2:2014cv00252 In the Eastern District of Texas Marshall Division
Retained by Heim, Payne & Chorush, L.L.P. for Packet Intelligence Technology
relates to secure networking.

Advanced Micro Devices, Inc. v. LG Corp., et al.
5:14-cv-01012 U.S. District Court for the Northern District of California Retained by Robins, Kaplan, Miller & Ciresi LLP on behalf of Advanced Micro Devices, Inc.
Technology relates to microprocessors and multimedia.

dunnhumby USA, LLC, et al. v. emnos USA Corp.
U.S. District Court for the Northern District of Illinois
Case No.: 1:13-cv-00399
Retained by Baker Hostetler who represents dunnhumby Technology
relates to databases.

Parthenon Unified Memory Architecture LLC, v. HTC CORPORATION and HTC AMERICA, INC., LG ELECTRONICS, INC. and LG ELECTRONICS USA, INC.
Case No. 2:14-cv-690-JRG-RSP
Case No. 2:14-cv-691-JRG-RSP
Eastern District of Texas, Marshall Division Technology
relates to multimedia systems.
Retained on behalf of Parthenon

Parthenon Unified Memory Architecture LLC, v. SAMSUNG ELECTRONICS CO., LTD., ET AL. HUAWEI TECHNOLOGIES CO. LTD, HUAWEI TECHNOLOGIES USA, INC. and HUAWEI DEVICE USA, INC. and MOTOROLA MOBILITY LLC
Case No. 2:14-cv-902-RSP
Case No. 2:14-cv-687-JRG-RSP
Case No. 2:14-cv-689-JRG-RSP
Eastern District of Texas, Marshall Division Technology
relates to multimedia systems.
Retained on behalf of Parthenon

NEXUS DISPLAY TECHNOLOGIES LLC, Plaintiff, v. DELL INC., LENOVO (UNITED STATES) INC., PANASONIC CORP. OF NORTH AMERICA, EIZO CORP. AND
EIZO INC., NEC CORP. AND NEC DISPLAY SOLUTIONS OF AMERICA, INC.
Civil Action No: 2:14-cv-00762
Civil Action No: 2:14-cv-00762
Civil Action No. 2:14-cv-00763
Civil Action No. 2:14-cv-00764
Civil Action No. 2:14-cv-00765
Technology relates to communications circuits.
Retained on behalf of Nexus Display Technologies

INTEL CORPORATION, v. FUTURE LINK SYSTEMS, LLC
C.A. No. 1:14-CV-00377-LPS
District Court for Delaware
Technology relates to communications circuits.
Retained on behalf of Future Link Systems

2014

Content Guard v. Apple Inc., Amazon.com Inc., BlackBerry Corporation, Huawei Device USA, Inc., Motorola Mobility LLC, BlackBerry Limited, HTC Corporation, HTC America, Inc., Huawei Technologies Co., Ltd., Samsung Electronics Co., Ltd., Samsung Electronics America, Inc., and Samsung Telecommunications America, LLC
The Eastern District of TX, case 2:13-cv-01112-JRG
Retained by McKool Smith on behalf of Content Guard
Technology relates to digital right management.

Content Guard v. Google Inc.
The Eastern District of TX, Case 2:14-cv-00061-JRG Retained
by McKool Smith on behalf of Content Guard Technology
relates to digital right management.

Solocron Media LLC v. Verizon Communications, Inc., Cellco Partnership d/b/a/
Verizon Wireless, AT&T Mobility LLC, T-Mobile USA, Inc., and Sprint Spectrum L.P.
The Eastern District of TX, Case 2:13-cv-1059-JRG
Retained by tensegrity on behalf of Solocron
Technology relates to MMS and multi-media transcoding.

Enterprise Systems Technologies S.a.r.l v. Apple Inc., Cirrus Logic Inc., HTC Corporation, LG Electronics, Inc., Samsung Electronics Co
The International Trade Commission of the United States of America
Inv. No. 337-TA-924
Retained by Mintz, Levin, Cohn, Ferris, Glovsky and Popeo LLP for complainant.
Technology relates to device drivers.

Ziilabs Inc., LTD., v. Samsung and Apple
Case 2:14-cv-00203 In the Eastern District of Texas Marshall Division
Retained by Davis of Heim, Payne & Chorush, L.L.P. for Ziilabs Technology
relates to graphics systems.

Packet Intelligence LLC v. Cisco
Case 2:2014cv00252 In the Eastern District of Texas Marshall Division
Retained by Heim, Payne & Chorush, L.L.P. for Packet Intelligence Technology
relates to secure networking.

Rockstar Consortium US v. Asustek, Google, HTC, LGE, Pantech, Samsung and ZTE
2:13-cv-00899 In the Eastern District of Texas Marshall Division
Retained by McKool Smith for Rockstar
Technology relates to user interfaces and messaging.

Advanced Micro Devices, Inc. v. LG Corp., et al.
5:14-cv-01012 U.S. District Court for the Northern District of California Retained by Robins, Kaplan, Miller & Ciresi LLP on behalf of Advanced Micro Devices, Inc.
Technology relates to microprocessors and multimedia.

Pragmatus Mobile LLC v. Samsung Electronics Co. Ltd, etc.
U.S. ITC Inv. No. 337-TA-905
Retained by Flagship IP. P.C. on behalf of Complainant Pragmatus Mobile LLC
Technology relates to converged devices.

CallCopy Inc.
Assistance with Reexamination efforts related to Verint Americas Inc.
Retained by Baker Hostetler
Technology related to call systems and call center management. Submitted declarations with the USPTO.

dunnhumby USA, LLC, et al. v. emnos USA Corp.
U.S. District Court for the Northern District of Illinois
Case No.: 1:13-cv-00399
Retained by Baker Hostetler who represents dunnhumby Technology
relates to database technology.

Graphics Properties Holdings, v. Panasonic et al.
ITC Inv. No. 337-TA-884
Retained by Mintz, Levin, Cohn, Ferris, Glovsky and Popeo LLP for complainant.
Provided testimony through expert reports, deposition and at trial. Technology relates to graphics hardware.

2013

Graphics Properties Holdings, v. Panasonic et al.
ITC Inv. No. 337-TA-884
Retained by Mintz, Levin, Cohn, Ferris, Glovsky and Popeo LLP for complainant.
Provided testimony through expert reports and deposition. Technology relates to graphics hardware.

CallCopy Inc.
Assistance with Reexamination efforts related to Verint Americas Inc.
Retained by Hahn Loeser & Parks LLP
Technology related to call systems and call center management. Submitted declarations with the USPTO.

dunnhumby USA, LLC, et al. v. emnos USA Corp.
U.S. District Court for the Northern District of Illinois
Case No.: 1:13-cv-00399
Retained by Baker Hostetler who represents dunnhumby Technology
relates to database technology.

Microlinc, LLC v. INTEL CORPORATION, ACER, INC., ACER AMERICA CORPORATION, ADVANCED MICRO DEVICES, INC., APPLE, INC., DELL, INC., GATEWAY, INC., HEWLETT-PACKARD COMPANY, LENOVO GROUP, LTD, LENOVO (UNITED STATES), INC., NVIDIA CORPORATION, SONY COMPUTER ENTERTAINMENT AMERICA INC., TOSHIBA CORPORATION, TOSHIBA AMERICA INC., and TOSHIBA AMERICA INFORMATION SYSTEMS, INC.
Eastern District of Texas
Retained by Eugene M. Cummings, P.C.
Submitted reports regarding infringement, invalidity and claim construction along with numerous declarations. Appeared at the Markman hearing. Deposed. Technology relates to microprocessor communications.

FlashPoint Technology, Inc. v. HTC Corporation, HTC America, Inc., Pantech Co., Ltd., Pantech Wireless, Inc., Huawei Technologies Co., Ltd., FutureWei Technologies, Inc. d/b/a Huawei Technologies (USA), ZTE Corporation, ZTE (USA) Inc.,
USA International Trade Commission
Certain Electronic Imaging Devices, Inv. No. 337-TA-850
Expert for FlashPoint, who is represented by Pepper Hamilton, LLP.
Submitted reports numerous declarations. Testified at the ITC. Deposed. Technology relates to converged devices.

2012

FlashPoint Technology, Inc. v. HTC Corporation, HTC America, Inc., Pantech Co., Ltd., Pantech Wireless, Inc., Huawei Technologies Co., Ltd., FutureWei Technologies, Inc. d/b/a Huawei Technologies (USA), ZTE Corporation, ZTE (USA) Inc.,
USA International Trade Commission
Certain Electronic Imaging Devices, Inv. No. 337-TA-850
Expert for FlashPoint, who is represented by Pepper Hamilton, LLP.
Submitted reports numerous declarations. Testified at the ITC. Deposed. Technology relates to converged devices.

Graphics Properties Holdings v. HTC et. al.
Graphics Properties Holdings v. LG Electronics et. al.
Graphics Properties Holdings v. Sony et. al.
Graphics Properties Holdings v. Samsung Electronics et. al.
Graphics Properties Holdings v. Apple
Graphics Properties Holdings v. Research In Motion et. al.
United States District Court for the District of Deleware
Testifying expert witness for GPH on matters related to invalidity.
Pepper Hamilton LLP and Mintz, Levin, Cohn, Ferris, Glovsky and Popeo LLP represent plaintiffs.
Subject matter relates to graphics systems and CPU microarchitecture. Submitted reports numerous declarations. Deposed.

IPValue
Non-litigation related patent evaluation services.

TV Interactive Data Corporation v. Sony Corporation, Sony Computer Entertainment Inc, Sony Computer Entertainment America Inc, Sony Corporation of America, Sony Electronics Inc, Samsung Electronics Co Ltd, Samsung Electronics America Inc, Royal Philips Electronics NV, Philips Electronics North America Corporation, Toshiba Corporation, Toshiba America Inc., Toshiba America Consumer Products LLC, Panasonic Corporation, Panasonic Corporation of North America, Victor Company of Japan, LTD, JVC Americas Corp, LG Electronics Inc, LG Electronics USA Inc., Zenith Electronics LLC, Pioneer Corporation, Pioneer Electronics (USA) Inc., Sharp Corporation, Sharp Electronics Corporation, Funai Electric Co., Ltd., Funai Corporation, Inc., D&M Holdings Inc., D&M Holdings US Inc. and Denon Electronics (USA), LLC
United States District Court for the Northern District of California Case 5:2010cv00475
Consultant for defendant Funai
Defendants Funai Electric Co. Ltd. and Funai Corporation, Inc. are represented by Kevin W. Kirsch, David A. Mancino, John F. Bennett, Matthew P. Hayden, and Hayes F. Michel of Baker Hostetler
Subject matter relates to DVDs and computer Operating Systems.

Interval Licensing LLC, v. AOL, Inc.,; Apple, Inc.; eBay, Inc.; Facebook, Inc.; Netflix, Inc.; Office depot, Inc.; Officemax, Inc.; Staples, Inc.; Yahoo! Inc.; and Youtube, LLC.
United States District Court for the Western District of Washington Case 2:10-cv-01385
Consultant for Interval Licensing LLC
Technology relates to user interfaces and data organization.

Interval Licensing LLC is represented by Michael F. Heim, Eric J. Enger, and Nathan J. Davis of Heim, Payne & Chorush, L.L.P.
Subject matter relates to online search, advertising and data presentation.

Submitted declarations related to claim construction. Deposed. Technology relates to user interfaces and data organization.

St. Clair Intellectual Property Consultants, Inc. v. Acer, Inc., Acer America Corporation, Dell Inc., Gateway Companies, Inc., Gateway, Inc., Lenovo Group, Limited and Lenovo (United States) Inc.
United States District Court for the District of Delaware - 1:2009cv00354
Consultant for St. Clair Intellectual Property Consultants
St. Clair is represented by R. Terrance Rader, Charles W. Bradley, Glenn E. Forbis, and Justin S. Cohen of Rader Fishman & Grauer PLLC
Subject matter relates to power management and graphics hardware.
Filed reports regarding validity and infringement and provided deposition testimony.

ST. CLAIR INTELLECTUAL PROPERTY CONSULTANTS, INC., v. APPLE, INC. HIGH TECH COMPUTER CORP., a/k/a HTC CORP., HTC (B.V.I.), HTC AMERICA, INC., EXEDEA, INC., RESEARCH IN MOTION LTD., and RESEARCH IN MOTION CORPORATION
United States District Court for the District of Delaware - 1:10-cv-00982-LPS
Consultant for St. Clair Intellectual Property Consultants
St. Clair is represented by R. Terrance Rader, Charles W. Bradley, Glenn E. Forbis, and Justin S. Cohen of Rader Fishman & Grauer PLLC Filed a declaration.
Subject matter relates to cameras and messaging.

Red Chalk Group
Subject matter relates to patent analysis.

Siemens
Subject matter relates to patent analysis.

Phoenix Technologies
Subject matter relates to patent analysis.

2011

TV Interactive Data Corporation v. Sony Corporation, Sony Computer Entertainment Inc, Sony Computer Entertainment America Inc, Sony Corporation of America, Sony Electronics Inc, Samsung Electronics Co Ltd, Samsung Electronics America Inc, Royal Philips Electronics NV, Philips Electronics North America Corporation, Toshiba Corporation, Toshiba America Inc., Toshiba America Consumer Products LLC, Panasonic Corporation, Panasonic Corporation of North America, Victor Company of Japan, LTD, JVC Americas Corp, LG Electronics Inc, LG Electronics USA Inc., Zenith Electronics LLC, Pioneer Corporation, Pioneer Electronics (USA) Inc., Sharp Corporation, Sharp Electronics Corporation, Funai Electric Co., Ltd., Funai

Corporation, Inc., D&M Holdings Inc., D&M Holdings US Inc. and Denon Electronics (USA), LLC
United States District Court for the Northern District of California
Case 5:2010cv00475
Consultant for defendant Funai
Defendants Funai Electric Co. Ltd. and Funai Corporation, Inc. are represented by Kevin W. Kirsch, David A. Mancino, John F. Bennett, Matthew P. Hayden, and Hayes F. Michel of Baker Hostetler
Subject matter relates to DVDs and computer Operating Systems.

Interval Licensing LLC, v. AOL, Inc.,; Apple, Inc.; eBay, Inc.; Facebook, Inc.; Netflix, Inc.; Office depot, Inc.; Officemax, Inc.; Staples, Inc.; Yahoo! Inc.; and Youtube, LLC.
United States District Court for the Western District of Washington Case 2:10-cv-01385
Consultant for Interval Licensing LLC
Interval Licensing LLC is represented by Michael F. Heim, Eric J. Enger, and Nathan J. Davis of Heim, Payne & Chorush, L.L.P.
Subject matter relates to online search, advertising and data presentation.

St. Clair Intellectual Property Consultants, Inc. v. Acer, Inc., Acer America Corporation, Dell Inc., Gateway Companies, Inc., Gateway, Inc., Lenovo Group, Limited and Lenovo (United States) Inc.
United States District Court for the District of Delaware - 1:2009cv00354
Consultant for St. Clair Intellectual Property Consultants
St. Clair is represented by R. Terrance Rader, Charles W. Bradley, Glenn E. Forbis, and Justin S. Cohen of Rader Fishman & Grauer PLLC Technology relates to power management.

ST. CLAIR INTELLECTUAL PROPERTY CONSULTANTS, INC., v. APPLE, INC. HIGH TECH COMPUTER CORP., a/k/a HTC CORP., HTC (B.V.I.), HTC AMERICA, INC., EXEDEA, INC., RESEARCH IN MOTION LTD., and RESEARCH IN MOTION CORPORATION
United States District Court for the District of Delaware - 1:10-cv-00982-LPS
Consultant for St. Clair Intellectual Property Consultants
St. Clair is represented by R. Terrance Rader, Charles W. Bradley, Glenn E. Forbis, and Justin S. Cohen of Rader Fishman & Grauer PLLC Filed a declaration.
Subject matter relates to power management and graphics hardware.

Negotiated Data Solutions
Consultant for Negotiated Data Solutions
Negotiated Data Solutions is represented by Susman, Godfrey LLP. Subject matter relates to IO devices.

Acqis LLC

Declarations filed with USPTO in response to a series of reexamination requests.
Deposed in a related case (Texas Eastern District Court 6:2009cv00148) as a consequence of declarations.
Acqis LLC is represented by Cooley, Godward, Kronish LLP Subject matter relates to computer systems.

FlashPoint Technology, Inc. v. Nokia Corp. of Finland, Nokia, Inc. of Irving, Texas, Research In Motion Ltd. of Canada, Research In Motion Corp. of Irving, Texas, HTC Corp. of Taiwan, HTC America, Inc. of Bellevue, Washington, LG Electronics of South Korea, LG Electronics U.S.A., Inc. of Englewood Cliffs, New Jersey, and LG Electronics MobileComm U.S.A., Inc. of San Diego, California
US ITC Inv. No. 337-TA-726
Consultant for FlashPoint Technology, Inc.
FlashPoint Technology Inc. is represented by William D. Belanger, Gregory D. Len, and Frank Liu of Pepper Hamilton LLP.
Filed reports regarding validity and infringement and provided testimony both via deposition and in court.
Subject matter relates to camera systems.

MicroUnity, Inc.v. Texas Instruments Inc., Qualcomm, Inc., Samsung Electronics Co LTD, Motorola, Inc., Nokia Corporation, Palm, Inc., Samsung TeleCommunications America, LLC, Acer, Inc., HTC Corporation, Google Inc., LG Electronics Inc, Apple, Inc., Cellco Partnership d/b/a Verizon Wireless, Sprint Nextel Corporation and AT&T
2:2010cv00091
Consultant for MicroUnity, Inc.
Eastern District of Texas
MicroUnity, Inc., is represented by Heim, Payne, and Chorush LLP. Subject matter relates to graphics hardware and microarchitecture.

2010

TV Interactive Data Corporation v. Sony Corporation, Sony Computer Entertainment Inc, Sony Computer Entertainment America Inc, Sony Corporation of America, Sony Electronics Inc, Samsung Electronics Co Ltd, Samsung Electronics America Inc, Royal Philips Electronics NV, Philips Electronics North America Corporation, Toshiba Corporation, Toshiba America Inc., Toshiba America Consumer Products LLC, Panasonic Corporation, Panasonic Corporation of North America, Victor Company of Japan, LTD, JVC Americas Corp, LG Electronics Inc, LG Electronics USA Inc., Zenith Electronics LLC, Pioneer Corporation, Pioneer Electronics (USA) Inc., Sharp Corporation, Sharp Electronics Corporation, Funai Electric Co., Ltd., Funai Corporation, Inc., D&M Holdings Inc., D&M Holdings US Inc. and Denon Electronics (USA), LLC
United States District Court for the Northern District of California

Case 5:2010cv00475
Consultant for defendant Funai
Defendants Funai Electric Co. Ltd. and Funai Corporation, Inc. are represented by Kevin W. Kirsch, David A. Mancino, John F. Bennett, Matthew P. Hayden, and Hayes F. Michel of Baker Hostetler
Subject matter relates to DVDs and computer Operating Systems.

Interval Licensing LLC, v. AOL, Inc.,; Apple, Inc.; eBay, Inc.; Facebook, Inc.; Netflix, Inc.; Office depot, Inc.; Officemax, Inc.; Staples, Inc.; Yahoo! Inc.; and Youtube, LLC.
United States District Court for the Western District of Washington Case 2:10-cv-01385
Consultant for Interval Licensing LLC
Interval Licensing LLC is represented by Michael F. Heim, Eric J. Enger, and Nathan J. Davis of Heim, Payne & Chorush, L.L.P.
Subject matter relates to online search, advertising and data presentation.

St. Clair Intellectual Property Consultants, Inc. v. Acer, Inc., Acer America Corporation, Dell Inc., Gateway Companies, Inc., Gateway, Inc., Lenovo Group, Limited and Lenovo (United States) Inc.
United States District Court for the District of Delaware - 1:2009cv00354
Consultant for St. Clair Intellectual Property Consultants
St. Clair is represented by R. Terrance Rader, Charles W. Bradley, Glenn E. Forbis, and Justin S. Cohen of Rader Fishman & Grauer PLLC
Subject matter relates to power management and graphics hardware.

Negotiated Data Solutions
Consultant for Negotiated Data Solutions
Negotiated Data Solutions is represented by Susman, Godfrey LLP. Subject matter relates to IO devices.

Acqis LLC
Declarations filed with USPTO in response to a series of reexamination requests.
Deposed in a related case (Texas Eastern District Court 6:2009cv00148) as a consequence of declarations.
Acqis LLC is represented by Cooley, Godward, Kronish LLP Subject matter relates to computer systems.

FlashPoint Technology, Inc. v. Nokia Corp. of Finland, Nokia, Inc. of Irving, Texas, Research In Motion Ltd. of Canada, Research In Motion Corp. of Irving, Texas, HTC Corp. of Taiwan, HTC America, Inc. of Bellevue, Washington, LG Electronics of South Korea, LG Electronics U.S.A., Inc. of Englewood Cliffs, New Jersey, and LG Electronics MobileComm U.S.A., Inc. of San Diego, California
US ITC Inv. No. 337-TA-726
Consultant for FlashPoint Technology, Inc.

FlashPoint Technology Inc. is represented by William D. Belanger, Gregory D. Len, and Frank Liu of Pepper Hamilton LLP.
Subject matter relates to camera systems.
Filed reports regarding validity and infringement and provided testimony both via deposition and in court.

MicroUnity, Inc.v. Texas Instruments Inc., Qualcomm, Inc., Samsung Electronics Co LTD, Motorola, Inc., Nokia Corporation, Palm, Inc., Samsung TeleCommunications America, LLC, Acer, Inc., HTC Corporation, Google Inc., LG Electronics Inc, Apple, Inc., Cellco Partnership d/b/a Verizon Wireless, Sprint Nextel Corporation and AT&T
2:2010cv00091
Consultant for MicroUnity, Inc.
Eastern District of Texas
MicroUnity, Inc., is represented by Heim, Payne, and Chorush LLP. Subject matter relates to graphics hardware and microarchitecture.

Saxon Innovations, LLC v. Apple, Inc., Gateway, Inc., Gateway Companies, Inc., Acer America Corp., Acer, Inc., Hewlett-Packard Co. and Dell, Inc
Eastern District of Texas 6:2008cv00265
Consultant for Saxon Innovations, LLC
Representing Saxon Innovations, LLC, who was represented by Heim, Payne, and Chorush LLP.
Subject matter relates to multiprocessors systems and computer systems. Filed reports regarding validity and infringement and provided testimony both via deposition and in court.

Joel M. Geddis and Shaun Legacy v. Microsoft Corporation, Toshiba Corporation, Freescale Semiconductor, Inc., Wolfson Microelectronics, Inc., Wolfson Microelectronics PLC
Superior Court of Washington in and for King County 08-2-40731-4 SEA
Consultant for Toshiba Corporation
Toshiba Corporation is represented by Forsberg & Umlauf, P.S. Subject matter relates to portable audio devices.

2009

Saxon Innovations, LLC v. LG Electronics Inc., LG Electronics USA Inc., LG Electronics Mobilecomm U.S.A., Inc., Samsung Electronics Co. Ltd, Samsung Electronics America Inc., Samsung Telecommunications America LLC, Palm Incorporated, Research in Motion Ltd, Research In Motion Corporation, Nintendo Co. Ltd, and Nintendo of America
Eastern District of Texas 6:07-CV-490
Prepared expert report regarding invalidity on behalf of Saxon Innovations, LLC.
Saxon Innovations, LLC, was represented by Pepper Hamilton LLP.

Subject matter relates to multiprocessors systems and computer systems. Filed reports regarding validity and infringement and provided testimony both via deposition and in court.

Saxon Innovations, LLC, v. Nokia Corp., Research In Motion Ltd., High Tech Computer Corp., Palm, Inc., Panasonic Corporation, US
ITC 337-TA-667
Expert witness for Saxon Innovations, LLC.
Submitted reports, was deposed, and testified at trial regarding claim construction and invalidity.
Saxon Innovations, LLC was represented by William D. Belanger, Aaron J. Levangie, Gregory D. Len, and Charles H. Carpenter of Pepper Hamilton LLP.
Subject matter relates to multiprocessors systems and computer systems.

Advanced Micro Devices, Inc. v. Samsung Electronics Co, Ltd.
U.S. District Court for the Northern District of California 3:08-cv-00986-SI Consultant for Advanced Micro Devices, Inc. Reverse engineering efforts to support infringement contentions.
Advanced Micro Devices, Inc is represented by Robins, Kaplan, Miller and Ciresi LLP.
Subject matter relates to memory devices.

Intellectual Ventures
Subject matter relates to patent analysis.

2005

Speedera Networks,Inc., v. Akamai Technologies,Inc.
U.S. District Court for the Northern District of California 5:02-cv-03050-JW Consultant for Speedera Networks, Inc.
Reviewed depositions and provided review of computer security.
Speedera Networks, Inc. was represented by Jim Bowman of O'Melveny & Myers LLP.
Technology related to computer security.

Research in Motion Limited et al. v. Inpro II Licensing SARL
U.S. District Court for the Northern District of Texas 3:03-cv-02669-B Consultant for InPro II Licensing SARL . Provided patent analysis.
InPro II Licensing SARL was represented by Dechert. Technology related to IO devices.

Digi International, Inc. v. Lantronix, Inc., 0:04-cv-01560-DWF-SRN U.S. District Court for the District of Minnesota
Consultant for Lantronix, Inc. Prepared expert reports.
Lantronix, Inc. was represented by Stradling Yocca Carlson & Rauth. Technology related to IO devices.

2004

Intergraph Hardware Technologies Company, Inc v. Hewlett-Packard
U.S. District Court for the Eastern District of Texas *Company*6:04-cv-00214-LED
Expert for Intergraph Hardware Technologies Company, Inc. Prepared expert reports and claim charts.
Intergraph represented by McDermott, Will and Emery LLP. Technology related to computer communications.

Mentor Graphics
Provided licensing support and IP valuation.
Technology related to emulation.

Veterinary Imaging Centers, Inc. v. Sound Technologies, Inc., et al., 1:02-cv-00282-DCN
U.S. District Court for the Northern District of Ohio
Consultant for Sound Technologies, Inc. Prepared expert reports. Technology related to computer communication.

Interealty Corp. v. Superaltive, Inc., et al., 8:01-cv-00969-AHS-AN
U.S. District Court for the Central District of California
Consultant for Superlative,Inc. Matter involved software copyright. Prepared expert reports and was deposed.
Superlative, Inc. was represented by Edward F. O'Connor with the law firm of Levin & O'Connor.
Technology related to databases.

All Computers, Inc. v. Intel Corporation, 1:04-cv-00586-GBL-LO
U.S. District Court for the Eastern District of Virginia
Consultant for All Computers, Inc. Filed numerous expert reports. Deposed by Ruffin Cordell of Fish & Richardson.
Plaintiff was represented by Edward F. O'Connor with the law firm of Levin & O'Connor.
Technology related to microprocessors.

2001

Ikos Systems, Inc. v. Axis Systems, Inc., 5:01-cv-21079-JW
U.S. District Court for the Northern District of California
Filed numerous expert reports. Deposed by Ikos Counsel. Presented tutorial to Judge Ware. Deposed.
Axis Sytems, Inc. was represented by George Riley of O'Melveny and Myers
Technology related to emulation.

Professional Activities

Member of ACM and IEEE
Program Chair 26'th International Symposium on Microarchitecture
General Chair 36'th International Symposium on Microarchitecture
Associate Editor of IEEE Transactions on Computers
Associate Editor of ACM Transactions on Embedded Computing Systems
Associate Editor of IEEE Computer
Program Committees: ISCA, MICRO, ISLPED, Network Processors Workshop, FPL, Complexity-Effective Design, RAW, Workshop on Mediaprocessors and DSP, FPT, INTERACT

Publications

Journal

1. W. H. Mangione-Smith, T.-P. Shih, S. G. Abraham, and E. S. Davidson, "Approaching a Machine-Application Bound in Delivered Performance on Scientific Codes," *Proceedings of the IEEE*, vol. 81, pp. 1166-1178, 1993.
2. C. Chien, S. Nazareth, P. Lettieri, S. Molloy, B. Schoner, W. A. I. Boring, J. Chen, C. Deng, W. H. Mangione-Smith, and R. Jain, "An Integrated Testbed for Wireless Multimedia Computing," *Journal of VLSI Signal Processing*, vol. 13, pp. 105-24, 1996.
3. K.-G. Chia, H. J. Kim, S. Lansing, W. H. Mangione-Smith, and J. Villasenor, "High Performance Automatic Target Recognition through Data-Specific VLSI," *IEEE Transactions on VLSI*, vol. 6, pp. 364-372, 1998.
4. N. R. Shnidman, W. H. Mangione-Smith, and M. Potkonjak, "On-line Fault Detection for Bus-Based Field Programmable Gate Arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, pp. 656-66, 1998.
5. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low Overhead Fault-Tolerant FPGA Systems," *IEEE Transactions on Very Large Scale Integration*, pp. 212-21, 1998.
6. D. Kirovski, C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Application-Driven Synthesis of Memory-Intensive Systems-on-Chip," *IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems*, vol. 18, pp. 1316-1326, 1999. 7. W. H. Mangione-Smith, "Technical Challenges for Designing Personal Digital Assistants," *Design Automation for Embedded Systems*, vol. 4, pp. 23-39, 1999.
8. D. Kirovski, J. Kin, and W. H. Mangione-Smith, "Procedure Based Program Compression," *International Journal of Parallel Programming*, vol. 27, pp. 457-475, 1999.
9. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Enhanced FPGA Reliability through Efficient Run-Time Fault Reconfiguration," *IEEE Transactions on Reliability*, vol. 49, pp. 296-304, 2000.

10. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based Watermarking Techniques for Design IP Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 1236-52, 2001.
11. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Fingerprinting Techniques for FieldProgrammable Gate Array Intellectual Property Protection," *IEEE Transactions on*
Computer-Aided Design of Integrated Circuits and Systems, vol. 20, pp. 1253-61, 2001.
12. C. Lee, J. Kin, M. Potkonjak, and W. H. Mangione-Smith, "Exploring Hypermedia Processor Design Space," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 27, pp. 171-186, 2001.
13. J. Kin, C. Lee, W. H. Mangione-Smith, and M. Potkonjak, "Exploring the Diversity of Multimedia Systems," *IEEE Transactions on VLSI Systems*, vol. 9, pp. 474-485, 2001.
14. J. Kin, M. Gupta, and W. H. Mangione-Smith, "Filtering Memory References to Increase Energy Efficiency," *IEEE Transactions on Computers*, vol. 49, pp. 1-15, 2002.
15. G. Memik and W. H. Mangione-Smith, “Precise Instruction Scheduling”, *To Appear in the Journal of Instruction-Level Parallelism*, 2005.

Conference

1. T. Conte and W. Mangione-Smith, "Determining Cost-Effective Multiple Issue Processor Designs," presented at International Conference on Computer Design, 1993.
2. W. H. Mangione-Smith, S. G. Abraham, and E. S. Davidson, "The Effects of Memory Latency and Fine-Grain Parallelism on Astronautics ZS-1 Performance," presented at Twenty-Third Hawaii International Conference on System Sciences, 1990.
3. W. H. Mangione-Smith, S. G. Abraham, and E. S. Davidson, "Architectural vs. Delivered Performance of the IBM RS/6000 and the Astronautics ZS-1," presented at Twenty-Fourth Hawaii International Conference on System Sciences, 1991.
4. W. H. Mangione-Smith, S. G. Abraham, and E. S. Davidson, "Vector Register Design for Polycyclic Vector Scheduling," presented at Fourth Conference on Architectural Support for Programming Languages and Operating Systems, 1991.
5. W. Mangione-Smith, S. G. Abraham, and E. S. Davidson, "Register Requirements of Pipelined Processors," presented at International Conference on Supercomputing, 1992.
6. W. H. Mangione-Smith and B. Hutchings, "Configurable Computing: The Road Ahead," presented at Reconfigurable Architectures Workshop, Geneva, 1997.
7. J. Kin, C. Lee, W. H. Mangione-Smith, and M. Potkonjak, "Hypermedia Processors: Design Space Exploration," presented at Multi-Media Signal Processing Conference, 1998.
8. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Fingerprinting Digital Circuits on Programmable Hardware," presented at Second International Workshop on Information Hiding, 1998.
9. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Signature Hiding Techniques for FPGA Intellectual Property Protection," presented at IEEE/ACM International Conference on Computer-Aided Design, 1998.

10. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "FPGA Fingerprinting Techniques for Protecting Intellectual Property," presented at Custom Integrated Circuits Conference, 1998.
11. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Efficiently Supporting FaultTolerance in FPGAs," presented at International Symposium on Field Programmable Gate Arrays, 1998.
12. W. Mangione-Smith, "Configurable Computing: Concepts and Issues," presented at Thirtieth Hawaii International Conference on System Sciences, 1997.
13. J. Villasenor, B. Schoner, C. Kang-Ngee, C. Zapata, K. Hea Joung, C. Jones, S. Lansing, and B. Mangione-Smith, "Configurable Computing Solutions for Automatic Target Recognition," presented at FPGAs for Custom Computing Machines, 1996.
14. B. Mangione-Smith, "Low Power Communications Protocols: Paging and Beyond," presented at IEEE Symposium on Low Power Electronics, 1995.
15. H. J. Kim, W. H. Mangione-Smith, and M. Potkonjak, "Protecting Ownership Rights of a Lossless Image Coder through Hierarchical Watermarking," presented at IEEE Workshop on Signal Processing Systems, 1998.
16. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking Techniques for Intellectual Property Protection," presented at Design and Automation Conference, 1998.
17. Rashid, J. Leonard, and W. H. Mangione-Smith, "Dynamic Circuit Generation for Solving Specific Problem Instances of Boolean Satisfiability," presented at FPGAs for Custom Computing Machines, 1998.
18. D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Synthesis of Power Efficient Systems-on-Silicon," presented at Asia and South Pacific Design Automation Conference, 1998.
19. J. Leonard and W. H. Mangione-Smith, "A Case Study of Partially Evaluated Hardware Circuits: Key-Specific DES," presented at Field-Programmable Logic and Applications, 1997.
20. N. R. Shnidman, W. H. Mangione-Smith, and M. Potkonjak, "Fault Scanner for Reconfigurable Logic," presented at Advanced Research in VLSI, 1997.
21. J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: an Energy Efficient Memory Structure," presented at International Symposium on Microarchitecture, 1997.
22. D. Kirovski, J. Kin, and W. H. Mangione-Smith, "Procedure Based Program Compression," presented at International Symposium on Microarchitecture, 1997.
23. D. Kirovski, C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Application-Driven Synthesis of Core-Based Systems," presented at International Conference on Computer Aided Design, 1997.
24. Mangione-Smith, P. S. Ghang, S. Nazareth, P. Lettieri, W. Boring, and R. Jain, "A Low Power Architecture for Wireless Multimedia Systems: Lessons Learned from Building a Power Hog," presented at International Symposium on Low Power Electronics and Design, 1996.
25. Lee, J. Kin, M. Potkonjak, and W. H. Mangione-Smith, "Media Architecture: General Purpose vs. Multiple Application-Specific Programmable Processor," presented at Design Automation Conference, 1998.

26. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a Tool for Evaluating and Synthesizing Multimedia and Communications Systems," presented at International Symposium on Microarchitecture, 1997.
27. Rashid, J. Asher, W. H. Mangione-Smith, and M. Potkonjak, "Hierarchical Watermarking for Protection of DSP Filter Cores," presented at IEEE 1999 Custom Integrated Circuits Conference, San Diego, CA USA, 1999.
28. H. Zou, H. J. Kim, S. Kim, B. Daneshrad, R. Wesel, and W. H. Mangione-Smith, "Equalized GMSK, Equalized QPSK, and COFDM, a Comparative Study for High Speed Wireless Indoor Data Communications," presented at International Vehicular Technology Conference, Houston, TX USA, 1999.
29. J. Kin, C. Lee, W. H. Mangione-Smith, and M. Potkonjak, "Power Efficient Mediaprocessors: Design Space Exploration," presented at Design Automation Conference, New Orleans, LA USA, 1999.
30. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Robust FPGA Intellectual Property Protection through Multiple Small Watermarks," presented at Design Automation Conference, 1999.
31. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Efficient Error Detection, Localization and Correction for FPGA-based Debugging," presented at Design Automation Conference, 2000.
32. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Algorithms for Efficient Runtime Fault Recovery on Diverse FPGA Architectures," presented at IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 1999.
33. Lee, J. Kin, M. Potkonjak, and W. Mangione-Smith, "Designing Power Efficient Hypermedia Processors," presented at International Symposium on Low Power Electronics and Design, 1999.
34. J. Lach, W. H. Mangione-Smith, and W. Potkonjak, "Enhanced Intellectual Property Protection for Digital Circuits on Programmable Hardware," presented at International Workshop on Information Hiding, 1999.
35. J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Runtime Logic and Interconnect Fault Recovery on Diverse FPGA Architectures," presented at Military and Aerospace Applications of Programmable Devices and Technologies, 1999.
36. Benyamin and W. Mangione-Smith, "Functional Unit Specialization through Code Analysis," presented at International Conference on Computer Aided Design, 1999.
37. J. Kin, C. Lee, W. H. Mangione-Smith, and M. Potkonjak, "A Technique for QoSBased System Partitioning," presented at ASP-DAC, 2000.
38. S. Brown, C. Lee, and W. H. Mangione-Smith, "Offline Program Re-Mapping to Improve Branch Prediction Efficiency in Embedded Systems," presented at ASP-DAC, 2000.
39. H. Kim and W. Mangione-Smith, "Factoring Large Numbers with Programmable Hardware," presented at International Symposium on Field Programmable Gate Arrays, 2000.
40. G. Memik, W. Hu, and W. Mangione-Smith, "NetBench: A Benchmarking Suite for Network Processor," presented at Computer Aided Design, 2001.

41. G. Memik, S. Memik, and W. Mangione-Smith, "Design and Analysis of a Layer Seven Network Processor Accelerator Using Reconfigurable Logic," presented at IEEE Symposium on FPGAs for Custom Computing Machines, 2002.
42. G. Memik and W. Mangione-Smith, "A Flexible Accelerator for Layer 7 Networking Applications," presented at Design Automation Conference, 2002.
43. Y. Cho, S. Navab, and W. Mangione-Smith, "Specialized Hardware for Deep Network Packet Filtering," presented at International Conference on Field Programmable Logic and Application, 2002.
44. G. Memik and W. Mangione-Smith, "Improving Power Efficiency of Multi-Core Network Processors Through Data Filtering," presented at International Conference on Compilers, Architecture and Synthesis for Embedded Systems, 2002.
45. G. Memik, G. Reinman, and W. Mangione-Smith, "Just Say No: Benefits of Early Cache Miss Determination," presented at International Symposium on High Performance Computer Architecture, 2002.
46. G. Memik and W. Mangione-Smith, "NEPAL: A Framework for Efficiently Structuring Applications for Network Processors," presented at NP-2: The Second Workshop on Network Processors, Anaheim, CA, 2003.
47. G. Memik, G. Reinman, and W. H. Mangione-Smith, "Reducing Energy and Delay Using Efficient Victim Caches," presented at International Symposium on Low Power Electronics and Design, Seoul, Korea, 2003.
48. Young Cho and William H. Mangione-Smith, “Programmable Hardware for Deep Packet Filtering on a Large Signature Set.” First Watson Conference on Interaction between Architecture, Circuits, and Compilers (P=ac 2), Yorktown, NY, Oct. 6-8, 2004
49. Young Cho and William H. Mangione-Smith, “Deep Packet Filter with Dedicated Logic and Read-Only-Memories.” IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa Valley / CA, April 2004.

Book Chapters

1. W. H. Mangione-Smith, "Register Requirements for High Performance Code Scheduling," in *The Interaction of Compiler Technology and Computer Architecture*, D. Lilja and P. Bird, Eds.: Kluwer Academic Press, 1994, pp. 51-84.
2. W. H. Mangione-Smith, "Performance Bounds for Rapid Computer System Evaluation," in *Fast Simulation of Computer Architectures*, T. Conte and C. Gimarc, Eds.: Kluwer Academic Press, 1995.
3. G. Memik and W. H. Mangione-Smith, "NEPAL: A Framework for Efficiently Structuring Applications for Network Processors," in *Network Processor Design: Issues and Practices*, vol. 2, P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk, Eds.: Morgan Kaufman, 2003.

Magazine Articles

1. W. H. Mangione-Smith, S. G. Abraham, and E. S. Davidson, "A Performance Comparison of the IBM RS/6000 and the Astronautics ZS-1," in *IEEE Computer*, vol. 24, 1991.
2. W. H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. K. Prasanna, and H. A. E. Spaanenburg, "Seeking Solutions in Configurable Computing," in *IEEE Computer*, vol. 30, 1997, pp. 38-43.
3. W. H. Mangione-Smith, "Application Design for Configurable Computing," in *IEEE Computer*, vol. 30, 1997, pp. 115-17.
4. J. Villasenor and W. H. Mangione-Smith, "Configurable Computing," in *Scientific American*, vol. 276, 1997, pp. 54-9.
5. W. H. Mangione-Smith, "Mobile Computing and Smart Spaces," in *IEEE Concurrency*, vol. 6, 1998, pp. 5-7.
6. W. H. Mangione-Smith, "'Beauty Depends on Size'-Aristotle," in *IEEE Concurrency*, vol. 6, 1998, pp. 8-10.