

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

Issue Date: June 24, 2014

Title: Graphics Processing Architecture Employing a Unified Shader

DECLARATION OF HANSPETER PFISTER, PH.D.
IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 8,760,454

TABLE OF CONTENTS

	Page
I. SUMMARY OF OPINION	8
II. EXPERT QUALIFICATIONS AND PRIOR TESTIMONY	8
III. MATERIALS REVIEWED	13
IV. UNDERSTANDING OF THE LAW	15
A. Legal Standards Relating to Claim Construction.....	15
B. Legal Standards Relating to Anticipation	16
C. Legal Standards Relating to Obviousness.....	18
V. TECHNOLOGY BACKGROUND.....	22
A. Graphics Processing	22
B. Parallel Processing	25
VI. THE '454 PATENT.....	28
VII. PROSECUTION HISTORY OF THE '454 PATENT.....	31
VIII. LEVEL OF ORDINARY SKILL IN THE ART.....	34
IX. GROUND 1: LINDHOLM '685 PATENT AND LINDHOLM '913 PATENT (CLAIMS 1-11).....	34
A. The Lindholm '685 Patent.....	35
B. The Lindholm '913 Patent.....	40
1. Combining the Lindholm Patents	45
2. Independent Claim 2	46
a. Claim 2 [preamble] – Unified Shader	46
b. Claim 2[a] – General Purpose Register Block	50
c. Claim 2[b] and [d] – Processor Unit.....	51
d. Claim 2[c] - Sequencer	54
3. Independent Claim 3	55
a. Claim 3 [preamble] – Unified Shader	56
b. Claim 3[a] and [c] – Processor Unit	56
c. Claim 3[b] – Shared Resources	62

TABLE OF CONTENTS

(continued)

	Page
4. Independent Claim 4	63
[a] a processor unit operative to perform vertex calculation operations and pixel calculation operations; and.....	63
a. Claim 4 [preamble] – Unified Shader	63
b. Claim 4[a] and [c] – Processor Unit	64
c. Claim 4[b] – Shared Resources	64
5. Independent Claim 5	64
a. Claim 5 [preamble] – Unified Shader	64
b. Claim 5[a] – Processor Unit	64
c. Claim 5[b] – Sequencer	64
6. Dependent Claim 6.....	64
7. Dependent Claims 7 and 10	65
8. Dependent Claim 8.....	66
9. Dependent Claim 9.....	66
10. Independent Claim 11	66
a. Claim 11 [preamble] – Unified Shader	67
b. Claim 11[a] and [c] – Processor Unit	67
c. Claim 11[b] – Instruction Store.....	67
11. Independent Claim 1	67
X. GROUND #2: CLAIMS 1-11 BASED ON THE COMBINATION OF AMANATIDES AND KOHN.....	69
A. Amanatides	70
B. The '454 Patent Does Not Claim a Multithreaded Graphics Processor	78
1. Combining Amanatides and Kohn.....	80
2. Independent Claim 2	81
a. Claim 2 [preamble] – Unified Shader	81

TABLE OF CONTENTS (continued)

	Page
b. Claim 2[a] – General Purpose Register Block	82
c. Claim 2[b] and [d] – Processor Unit.....	84
d. Claim 2[c] - Sequencer	88
3. Independent Claim 3	89
a. Claim 3 [preamble] – Unified Shader	89
b. Claim 3[a] and [c] – Processor Unit	89
c. Claim 3[b] – Shared Resources	93
4. Independent Claim 4	94
a. Claim 4 [preamble] – Unified Shader	94
b. Claim 4[a] and [c] – Processor Unit	94
c. Claim 4[b] – Shared Resources	94
5. Independent Claim 5	94
a. Claim 5 [preamble] – Unified Shader	94
b. Claim 5[a] – Processor Unit	94
c. Claim 5[b] – Sequencer	95
6. Dependent Claim 6.....	95
7. Dependent Claims 7 and 10	95
8. Dependent Claim 8.....	97
9. Dependent Claim 9.....	97
10. Independent Claim 11	97
a. Claim 11 [preamble] – Unified Shader	97
b. Claim 11[a] and [c] – Processor Unit	98
c. Claim 11[b] – Instruction Store.....	98
11. Independent Claim 1	98
XI. GROUND #3: CLAIMS 1-11 BASED ON THE COMBINATION OF SELZER AND FISKE.....	100
A. Selzer	100

TABLE OF CONTENTS

(continued)

	Page
B. Fiske	105
C. Combining Selzer and Fiske	109
1. Independent Claim 2	112
a. Claim 2 [preamble] – Unified Shader	112
b. Claim 2[a] – General Purpose Register Block	114
c. Claim 2[b] and [d] – Processor Unit.....	116
d. Claim 2[c] - Sequencer	118
2. Independent Claim 3	120
a. Claim 3 [preamble] – Unified Shader	120
b. Claim 3[a] and [c] – Processor Unit	120
c. Claim 3[b] – Shared Resources	121
3. Independent Claim 4	121
a. Claim 4 [preamble] – Unified Shader	121
b. Claim 4[a] and [c] – Processor Unit	121
c. Claim 4[b] – Shared Resources	121
4. Independent Claim 5	121
a. Claim 5 [preamble] – Unified Shader	121
b. Claim 5[a] – Processor Unit	121
c. Claim 5[b] – Sequencer	122
5. Dependent Claim 6.....	122
6. Dependent Claims 7 and 10	122
7. Dependent Claim 8.....	124
8. Dependent Claim 9.....	124
9. Independent Claim 11	124
a. Claim 11 [preamble] – Unified Shader	124
b. Claim 11[a] and [c] – Processor Unit	124
c. Claim 11[b] – Instruction Store.....	125

TABLE OF CONTENTS
(continued)

	Page
10. Independent Claim 1	125
XII. DECLARATION	127

I, Dr. Hanspeter Pfister, Ph.D., declare as follows:

1. I have been retained by counsel for Petitioner, Realtek Semiconductor Corp., as an expert in this *inter partes* review to examine whether claims 1-11 of U.S. Patent No. 8,760,454 (the '454 patent or Patent in Suit) are patentable over certain prior art.

2. I have personal knowledge of all the facts set forth herein, and if called to testify at any hearing in this *inter partes* review, I would competently testify and verify that testimony contained herein.

3. I am being compensated at my hourly rate of \$650 per hour. I am also being reimbursed for out-of-pocket expenses. My compensation does not depend in any way on the outcome of this Investigation or the particular opinions I express, or the testimony I give.

4. I expect to be available for deposition and to testify at the evidentiary hearing in this *inter partes* review to the extent required.

5. 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 scope and terms of my engagement for this declaration; 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 Patent in Suit; an analysis of the level of ordinary skill in the art related to the Patent in Suit; an analysis of the asserted references; and a patentability analysis of the challenged claims.

6. 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 review. 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 and/or other opinions provided by Petitioner's expert(s), or in light of any relevant orders from the Patent Trial and Appeal Board or other authoritative body.

I. SUMMARY OF OPINION

7. It is my opinion that claims 1-11 of the '454 Patent are at least rendered obvious by the prior art references discussed below. I will explain below my analysis and basis for my opinion in detail.

II. EXPERT QUALIFICATIONS AND PRIOR TESTIMONY

8. My curriculum vitae is attached hereto as Ex. 1004, which provides an accurate identification of my relevant background and experience in multi-threaded

graphics processors. For the past 25 years, I have focused on computer graphics processing in both industrial and academic settings. I have designed cutting-edge graphics processing systems, including the world's first real-time volume rendering graphics card. I have pioneered research into various aspects of computer graphics and received academic awards and industry recognition for my work in computer graphics. I am a Fellow of the ACM (Association of Computing Machinery), the IEEE (Institute of Electrical and Electronics Engineers), and a member of the ACM SIGGRAPH and the IEEE Visualization Academy.

9. In 1991, I received my M.Sc. in Electrical Engineering from the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. I then studied computer science at the State University of New York at Stony Brook, Stony Brook, NY, where I earned my M.S. in Computer Science in 1994 and my Ph. D. in Computer Science in 1996.

10. I am the Academic Dean of Computational Science and Engineering and An Wang Professor of Computer Science at the Harvard John A. Paulson School of Engineering and Applied Sciences at Harvard University. I am also an affiliate faculty member of the Harvard Center for Brain Science. My research in visual computing lies at the intersection of visualization, computer graphics, and computer vision. It spans a wide range of topics, including bio-medical visualization, image and video analysis, augmented and virtual reality, and data

science. From 2013 to 2017, I was director of the Institute for Applied Computational Science. For most of this work, I am leveraging the power of graphics processing units (“GPUs”) for computer graphics, high-throughput computing, and visualization. I am currently advising and co-advising six Ph. D. students, five post-doctoral fellows, and one post-doctoral research scientist. I also supervised numerous research and thesis projects and student interns.

11. Before joining Harvard, I worked for over a decade at Mitsubishi Electric Research Laboratories as Associate Director and Senior Research Scientist. I was the chief architect of Mitsubishi Electric's VolumePro, the world's first PC graphics card for real-time visualization of volume data. The technology was subsequently acquired by TeraRecon and is still in wide commercial use in medicine, biology, engineering, and oil and gas exploration. VolumePro received several technical awards, including Mitsubishi Electric President's Award in 2000. Since then, I have developed several new methods for high-quality and interactive volume visualization on GPUs. I received the 2010 IEEE Visualization Technical Achievement award in recognition of my “seminal technical achievements in real-time volume rendering.”

12. I am a pioneer in point-based computer graphics, a subfield of computer graphics that deals with modeling and rendering of point-sampled (e.g., laser-scanned) objects. I am among the first to introduce data-driven approaches

for complex real-world objects to computer graphics, including human faces. And I developed new camera and display technologies, including the world's first end-to-end real-time 3D TV system with auto-stereoscopic display. Currently, I am collaborating with Harvard scientists on novel analysis and visualization approaches in computational science, including neuroscience, genomics, systems biology, astronomy, and medicine. I am co-inventor of over 50 US patents and co-author on more than 200 peer-reviewed publications, including over 25 ACM SIGGRAPH papers, the premier forum in Computer Graphics. According to Google Scholar, my publications have been cited about 34,000 times, with about half of these citations since 2015, and I have an h-index of 89.

13. From 1999 to 2007, I built a rich academic program in computer graphics at the Harvard Extension School through the development of several new courses, including CSCI E-234 "Introduction to Computer Graphics" (1999-2007), CSCI E-235 "Advanced Computer Graphics" (2002), CSCI E-236 "Advanced Topics in Computer Graphics" (2004-2006), and INDR E-399 "Independent Study in Advanced Computer Graphics" (2003). After joining Harvard in 2007, I introduced several new undergraduate and graduate courses that are also offered to the public through the Harvard Extension School, including CS171 "Visualization" (2007-present), CS175 "Introduction to Computer Graphics" (2010), CS264 "Massively Parallel Computing" (2009-2011), CS205 "Introduction to

Computational Science" (2010-2012), and CS109 "Data Science" (2013-2018), and CS10 "Elements of Data Science" (2020-2022).

14. Between 2009 and 2016, in collaboration with Prof. Alan Aspuru-Guzik of the Chemistry and Chemical Biology faculty at Harvard, I was the co-PI (Principal Investigator) of Harvard's CUDA Center of Excellence (CCOE). The CCOE supported "outstanding research taking place in Massively Parallel Programming and Computing" using GPUs.

15. I have served on the papers committees of all major visualization and graphics conferences, including ACM SIGGRAPH, IEEE Visualization, EuroVis, Eurographics, Pacific Graphics, and many others. I have co-organized various international workshops, symposia, and conferences in computer graphics and visualization, including general conference chair of IEEE Visualization 2002 and technical papers chair of ACM SIGGRAPH 2012. I previously chaired and am currently a director of the IEEE Visualization and Graphics Technical Committee. In 2020, I was elected to the ACM SIGGRAPH Executive Committee as a director-at-large. I served on the editorial board of the IEEE Transactions on Visualization and Computer Graphics and the ACM Transaction on Graphics, the top two journals in the field. I am co-editor of the first textbook on Point-Based Computer Graphics, published by Elsevier in 2007, and the 2006 NIH/NSF Visualization Research Challenges Report. In addition to being a Fellow of the

ACM and the IEEE, I am a member of ACM SIGGRAPH and the Eurographics Association. I received the IEEE Golden Core Award, the IEEE Meritorious Service Award, and the Petra T. Shattuck Excellence in Teaching Award. I was elected to the IEEE Visualization Academy and the ACM SIGGRAPH Academy. The academies are an honorary group of individuals who have made substantial contributions to the fields of visualization and computer graphics, respectively.

16. I have been working as an independent consultant since 2007, specializing in computer graphics, visualization, data science, technical due diligence reviews, and in the provision of expert witness services, particularly patent infringement. My clients range from domestic start-ups to international Fortune 500 companies, and include Adobe, Disney Research, Novartis, as well as NVIDIA.

III. MATERIALS REVIEWED

17. I have reviewed the following materials in forming my opinions:

- U.S. Patent No. 8,760,454 (the “’454 Patent”)
- Prosecution history of the ’454 Patent and related patents
- U.S. Patent No. 7,038,685 (“Lindholm ’685 patent”) was filed on June 30, 2003 and issued on May 2, 2006. Ex. 1005. It is therefore, prior art to the ’454 patent under at least 35 U.S.C. § 102(e).

- U.S. Patent No. 7,015,913 (“Lindholm ’913 patent”) was filed on June 27, 2003 and issued on March 21, 2006. Ex. 1006. It is therefore, prior art to the ’454 patent under at least 35 U.S.C. § 102(e).
- John Amanatides and Edward Szurkowski, A Simple, Flexible, Parallel Graphics Architecture, In Proceedings of Graphics Interface at 155-160 (Canadian Information Processing Society 1993) (“Amanatides”) was published in Proc. Graphics Interface ’93 in May 1993. Ex. 1007. It is therefore prior art to the ’454 Patent under at least 35 U.S.C. § 102(a) and (b).
- Les Kohn and Neal Margulis, Introducing the Intel i860 64-bit Microprocessor, IEEE, Volume 9, Issue 4, pages 15-30, August 1989. Ex. 1008. It is therefore prior art to the ’454 patent under at least 35 U.S.C. § 102(a) and (b).
- Harald Selzer, Dynamic Load Balancing within a High Performance Graphics System, In Proceedings of Rendering, Visualization and Rasterization Hardware (Eurographics' 91 Workshop) at 37-53 (Springer-Verlag 1993) (“Selzer”) was published in 1993. Ex. 1009. It is therefore prior art to the ’454 patent under at least 35 U.S.C. § 102(a) and (b).

- Stuart Fiske and William J. Dally, Thread Prioritization: A Thread Scheduling Mechanism for Multiple-Context Parallel Processors, In Proceedings of First Symposium on High-Performance Computer Architecture, 1995 at 210-221 (IEEE 1995) (“Fiske”) was published in 1995. Ex. 1010. It is therefore prior art to the ’454 patent under at least 35 U.S.C. § 102(a) and (b).

18. I have tried my best to list all the materials I considered and reviewed. I may have accidentally left some materials off the above list that I cited in my analysis below. Such materials, if any, should be included in the above list of materials I considered and reviewed. I reserve the right to revise this list.

IV. UNDERSTANDING OF THE LAW

19. I am not a legal expert. In forming my opinions, I have been informed of and applied the legal standards as follows. I understand that the legal standards relating to anticipation and obviousness apply to the ’454 Patent based on its effective filing date of November 20, 2003.

A. Legal Standards Relating to Claim Construction

20. I understand that, during an *inter partes* review proceeding, patent claims are given their “broadest reasonable interpretation in light of the specification of the patent.” It is my further understanding that the prosecution history is relevant

to determining the correct construction of claim terms and that extrinsic evidence also may be relevant to establish the meaning of terms to the extent it is consistent with the specification and prosecution history.

B. Legal Standards Relating to Anticipation

21. I understand that a patent claim is “anticipated” and therefore invalid when a single prior art document describes, or a prior art product includes, every element recited, either expressly or inherently, in the claim. I further understand that, to be considered anticipatory, a written prior art reference must be enabling and describe the invention of the claim sufficiently to have placed it in possession of a person of ordinary skill in the field of the invention.

22. I understand that a prior art reference may anticipate a claim without disclosing an element of the claim if that missing element is necessarily present, or inherent, in the single anticipating reference as would be understood by one of ordinary skill in the art. I understand that inherency cannot be established by probabilities or possibilities; the mere fact that a certain thing may result from a given set of circumstances is not sufficient.

23. I understand that a patent claim is invalid if the claimed invention “was known or used by others in this country ... before the invention thereof by the [patent] applicant.” 35 U.S.C. § 102(a). I understand that, to be invalidating, knowledge or use must be accessible to the public. I understand that whether use or

knowledge is “public” is a fact-intensive question, the answer to which depends on such factors as whether some or all of the prior use or knowledge was maintained in confidence or as a trade secret, the existence of non-disclosure agreements or other confidentiality obligations imposed on members of the public who observed the use, the number of people involved, and the extent to which the use of the product informs an observer about the features of the claimed invention.

24. I understand that a patent claim is invalid if the claimed invention was “patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for patent” (35 U.S.C. § 102(a)) or the claimed 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)). I understand that the date of “the invention . . . by the applicant” in both § 102(a) and § 102(e) refers to the conception date of the claimed invention by the patent applicant.

25. I understand that a reference does not qualify as prior art under § 102(a) or § 102(e) if (1) the inventor conceived of the invention prior to the effective date of the reference and showed reasonable diligence in reducing the invention to practice from a time just prior to the effective date of the reference or (2) the inventor reduced the invention to practice prior to the effective date of the reference. I understand that conception requires an inventor to form in his or her mind a definite and particular

idea of a complete and operative invention, including every feature of the claim at issue. I have further been informed that there are two types of reduction to practice: actual reduction to practice (i.e., when all limitations of the claimed invention are embodied in a physical or tangible form that has been sufficiently tested to demonstrate it works for its intended purpose) and constructive reduction to practice (i.e., by the filing of a patent application).

26. I understand that a patent claim is invalid if the claimed “invention was patented or described in a printed publication in this or a foreign country . . . more than one year prior to the date of the application for patent in the United States” (35 U.S.C. § 102(b)). I understand that, to qualify as prior art, a printed publication must be shown to be accessible to the public (i.e., disseminated or otherwise made available to the extent that persons interested and ordinarily skilled in the subject matter or art, exercising reasonable diligence, can locate it).

C. Legal Standards Relating to Obviousness

27. I understand that a patent claim is “obvious” and therefore invalid when the differences between the claimed subject matter and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains.

28. In making an obviousness determination, I understand that there are several factors to consider: (1) the scope and content of the prior art; (2) the level of ordinary skill in the art at the time the invention was made; (3) the differences between the claimed invention and the prior art, if any, and (4) objective considerations, if any exist, such as any commercial success, copying, prior failure by others, licenses, longstanding need, and unexpected results.

29. I understand that prior art used to show that a claimed invention is obvious must be “analogous art.” Prior art is analogous art to the claimed invention if (1) it is from the same field of endeavor as the claimed invention (even if it addresses a different problem) or (2) the reference is reasonably pertinent to the problem faced by the inventor (even if it is not from the same field of endeavor as the claimed invention).

30. I understand that a patent claim composed of several elements is not proved obvious merely by demonstrating that each of its elements was independently known in the prior art, whether those references were set forth in a single reference or in a combination of references. Instead, it must have been obvious to arrange those elements in the same way as the elements in the claim.

31. I understand a single reference may be the basis for finding that a claim is obvious. When the obviousness determination relies on the combination of two or more references, however, the patent challenger must show that there is a

reason, suggestion, or motivation that would lead one of ordinary skill in the art to combine prior art references.

32. I understand that there is no single way to define the line between true inventiveness on the one hand (which is patentable) and the application of common sense and ordinary skill to solve a problem on the other hand (which may not be patentable). For example, market forces or other design incentives may be what produced a change in the technology of the prior art, rather than true inventiveness. I further understand that explicit teachings or suggestions in the prior art to make the modification or combination of elements claimed in the patent may be evidence of obviousness. Further, an innovation may be obvious if it applies a known technique that had been used to improve a similar device or method in a similar way. I understand that a claimed invention may be found to be obvious to try if the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those skilled in the art.

33. However, I also understand that it is improper to engage in hindsight when trying to determine the obviousness of a patent claim. Many true inventions might seem obvious with the benefit of hindsight. I understand that the obviousness inquiry must be conducted from the standpoint of a person of ordinary skill in the field at the time the claimed invention was made. What is known today, and what is learned from the teachings and disclosures of the patent itself

containing the claim under analysis, should not be considered. Nor should one use the patent claim as a guide to picking out elements of the prior art for combination.

34. I have been informed that various “secondary considerations” (sometimes referred to as objective indicia of non-obviousness) may support a determination of non-obviousness and that such secondary considerations must be considered as part of an obviousness analysis. I have been informed that secondary considerations of nonobviousness may include:

- commercial success of a product due to the merits of the claimed invention;
- a long felt need for the solution provided by the claimed invention;
- unsuccessful attempts by others to find the solution provided by the claimed invention;
- copying of the claimed invention by others;
- unexpected and superior results from the claimed invention; and
- acceptance by others of the claimed invention as shown by praise from others in the field or from the licensing of the claimed invention.

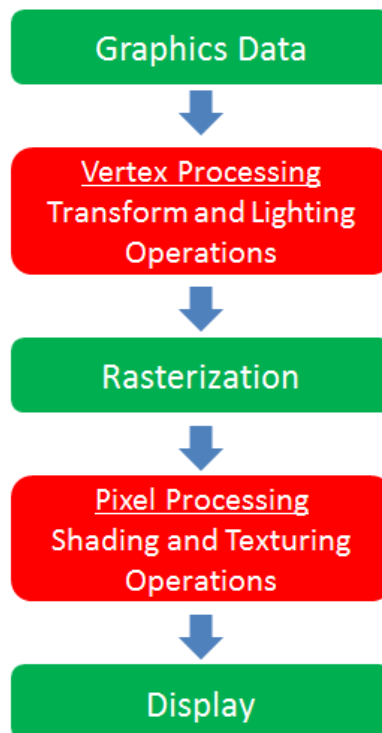
35. I understand that in order for such “secondary considerations” evidence to be relevant to the obviousness inquiry, there must be a relationship or “nexus” between the advantages and features of the claimed invention and the evidence of secondary considerations.

36. I understand that if a claim element is not obvious, then the claims that depend from it are not obvious.

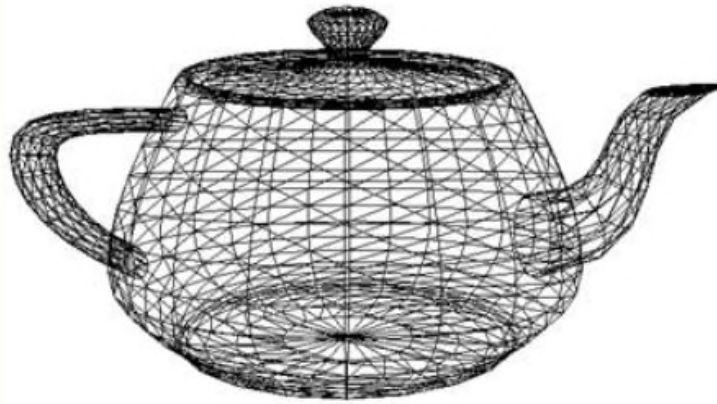
V. TECHNOLOGY BACKGROUND

A. Graphics Processing

37. Graphics processing has been an important part of any computer system for several decades. The purpose of a **graphics processor** is to generate complex shapes and structures to be displayed on a screen. To accomplish that purpose, the graphics processor converts a 3D object or scene (comprised of points in 3D space called **vertices** that make up shapes called **primitives**) into a 2D image to be displayed on a computer screen (comprised of **pixels**). Generally, 3D graphics processing starts with creating a mathematical model of each object. The model is then processed through a series of steps, referred to as a **graphics processing pipeline**, that renders the scene as a 2D image on a display:



38. In most cases, 3D objects are conceptualized as a series of triangles that cover the surface of an object, such as a teapot:

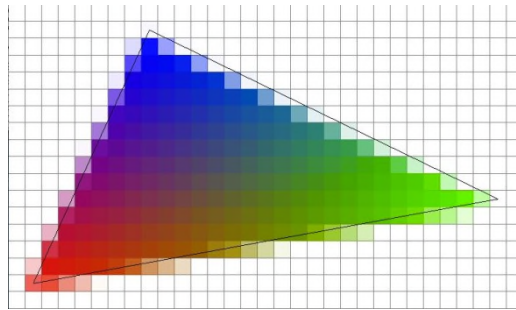


39. Each point of the primitive is called a **vertex**, and each vertex has certain **properties** represented as data. For example, a vertex includes not just its location but may also include other information, such as the object's color and its material properties (e.g., whether it is reflective). **Vertex processing** performs the steps in the graphics pipeline that transform these vertices from 3D space into 2D space and determines how lighting and other conditions in the 3D scene impact the color of the vertices. The operations used during vertex processing are commonly called **vertex shading**.

40. Vertices are assembled into primitives, typically triangles, by the **primitive processing** step. At this stage, primitives may also be generated computationally inside the graphics processing pipeline. During the primitive processing step, the graphics processor may perform additional operations on the

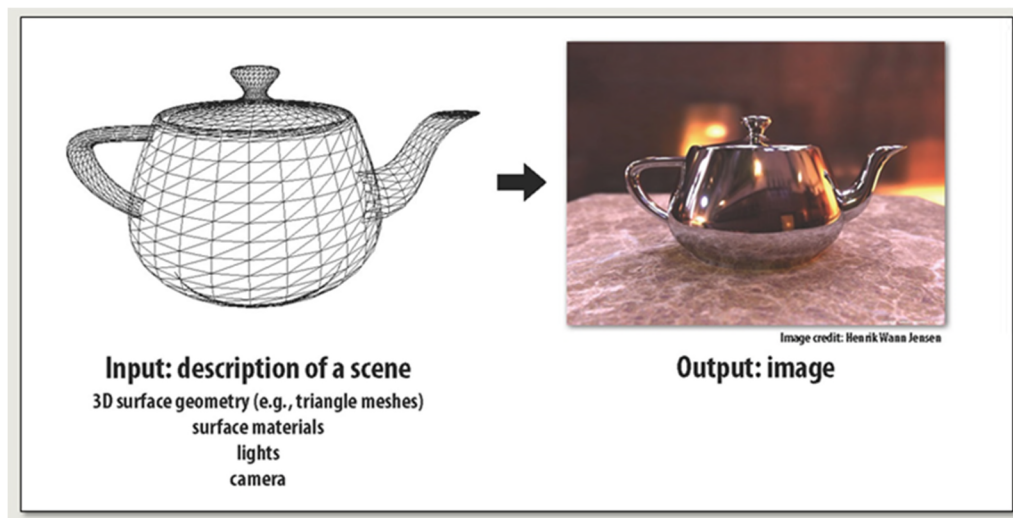
primitives, such as applying **texture mapping** and other shading effects. Texture mapping is an important step in the graphics pipeline because it allows the graphics processor to generate a realistic, high-quality image from the 3D model.

41. After the vertex processing, a step called **rasterization** determines what pixels on the 2D screen are covered by each primitive. At least one



fragment is generated for each pixel on the screen (as a result, the terms “fragment” and “pixel” are sometimes used interchangeably). Rasterization commonly includes the step of **scan conversion**, which involves stepping through the geometry of the primitives to determine which pixels are covered.

42. The initial values of each fragment (such as its color) are calculated using the vertex data and interpolation. **Pixel processing** then performs additional operations on the fragments or pixel data, including determining each pixel's final color. These are called **pixel shading** operations and may involve lighting, texture and bump mapping, translucency, and other phenomena. All this information is gathered by merging or blending pixels using **frame buffer operations** to display the final image on a screen.



B. Parallel Processing

43. To meet the high computational requirements, graphics processing uses different forms of **parallel processing**. A graphics processor has, at a minimum, the basic components shown in the following diagram.



44. First, an **instruction fetch** unit may retrieve the instructions that specify what needs to be processed. Second, there must be **registers** to store the

program's current data. Read registers store data being fetched from outside the processing core, whereas write registers store data that has been processed and will be written back to external memory. Finally, one or more **Arithmetic Logical Units (ALUs)** perform the necessary arithmetic and logic operations. A collection of ALUs, as shown in the figure above, may constitute a **SIMD (Single Instruction Multiple Data)** processing unit. Most SIMD units use some form of **vector instructions**, which expose single instructions for vector operations to programs.

45. Graphics processors may also execute multiple **threads** of instructions to further improve performance. A **thread** is a sequential set of instructions that specify what operations to perform, such as adding two values, accessing memory, comparing values with each other, etc. In a **single-threaded** processor, these instructions comprise a complete program, for example, a computer game or a multimedia application. In **multi-threaded** processors, a graphics program consists of multiple threads, for example, one for processing vertex data and one for processing pixel data. Each thread in a program will have its own thread **context**, such as constant values and state data stored in registers or pointers to graphics data in memory that are relevant for processing the thread.

46. Vertex data and pixel data both need to be processed. Since the operations for processing vertices differ from those used to process pixels, the prior

art used separate vertex and pixel processors. Dedicated **vertex processors** could only perform operations on vertex data, and dedicated **pixel processors** could only operate on pixel data. Separate vertex and pixel processors could not adapt to the varying needs of each image, some of which have a lot of vertex data to process while others have more pixel data. This was inefficient, as either vertex processors or pixel processors (sometimes called **shaders**) were often idle while the other type of processor became backlogged.

47. Before the '454 patent was filed, Erik Lindholm, the inventor on both the '685 and '913 prior art patents, recognized an issue in the prevailing approach that employed distinct, dedicated processors for vertex and pixel data. He understood this was inefficient as the data quantity of vertex and pixel varied across different images, but the system's processing capacity remained constant due to hardware constraints.

48. The requirement was for a method to handle vertex and pixel data through one processor. This processor should be able to process either vertex or pixel data and balance a shared pool of resources between the two forms of data. Erik Lindholm invented a **multi-threaded processor** capable of handling vertex and pixel data, also known as a **unified shader**. The invention further allowed for dynamic adjustment of the number of threads processing vertices or pixels based on available resources during operation. When switching threads (i.e., the

processor switches between vertex and pixel shader programs), the processor must save its current state and data to memory for the thread context and load another state and data from memory into the same thread context. This is also known as **context switching**. A **sequencer** is typically used to schedule the execution of threads and to manage context switches.

VI. THE '454 PATENT

49. The '454 patent ostensibly resolves an issue in the prior art requiring individual vertex and pixel shaders. Ex. 1001 at 2:12-29. However, it is important to note that this problem was addressed a decade earlier in the prior art, as at least Selzer and Amanatides recognized the efficiencies to be gained by processing both vertices and pixels on the same processor unit. In addition, the Lindholm patents included both vertex and pixel processing on the same multithreaded processing unit. All of these prior art references introduced the concept of a unified shader many years prior to the '454 patent.

50. The '454 patent, like the Lindholm patents, identifies that using separate hardware for vertex and pixel shaders leads to a larger processor footprint. *Id.* As such, the '454 patent introduces the utilization of a single hardware component, referred to as a “unified shader,” to execute both vertex and pixel shader programs. *Id.* at 2:58-3:3. The patent further defines a unified shader as a shader “configured to perform both vertex and pixel operations.” *Id.* at 3:10-12.

51. Figure 4A of the '454 patent is illustrative of the unified shader.

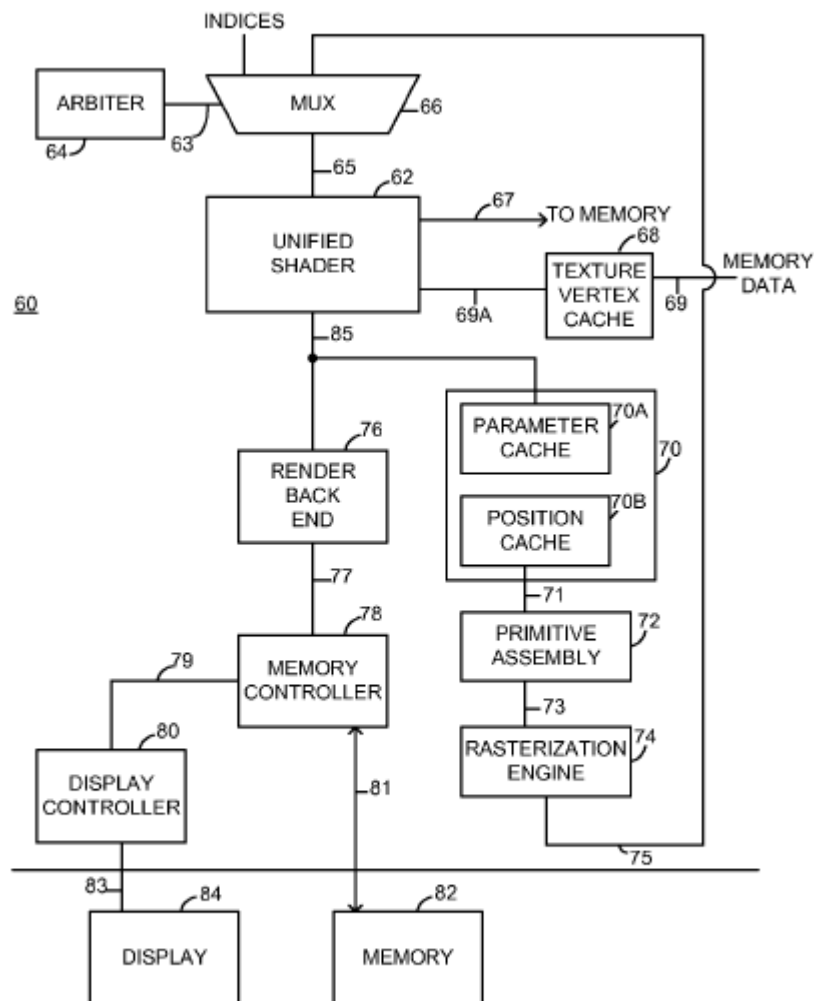


FIG. 4A

52. The graphics processor of the '454 patent encompasses an arbiter circuit, whose function is “selecting one of a plurality of inputs for processing in response to a control signal...” *Id.* at 4:19-39. The selected inputs are either vertex or pixel data. *Id.* at 2:62-3:3. Based on the selected input, the unified shader performs either a vertex or a pixel operation on the data. *Id.*

53. Within the unified shader is a general-purpose register block for storing vertex and pixel data to be processed. *Id.* at 4:29-39. The specification discloses that there was nothing inventive about the hardware comprising the “unified shader” itself, which is described as being “a processor (e.g., CPU) 96.” *Id.* at 4:30-34. “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).” *Id.* at 4:40-51.

54. In one embodiment, processor 96 is “logically partitioned into two sections,” configured to perform 32-bit floating point arithmetic operations and scalar operations (e.g., log, exponent, reciprocal square root). *Id.* It was known to a POSITA that vertex processing typically required floating point operations, and pixel processing required scalar operations. The processor unit within the unified shader can process vertex data concurrently with pixel data and alternate between the two types of operations. *Id.* at 5:32-36; 6:36-41.

55. Within the unified shader, the unit that chooses the next instruction for execution is called the sequencer. Ex. 1001 at 4:52-5:18. This determination might be made based on whether the general-purpose registers have enough available space to store incoming vertex data, and if not, pixel operations are continued. *Id.* at 5:41-52.

VII. PROSECUTION HISTORY OF THE '454 PATENT

56. Based on my review of the prosecution history, I understand that the patent application that led to the '454 patent was filed on May 17, 2011, and issued on June 14, 2014. The '454 patent claims priority to an application filed November 20, 2003.

57. On July 21, 2011, Examiner first issued a Non-Final Rejection for all claims as being anticipated by U.S. Patent No. 7,038,685 ("Lindholm"). Ex. 1002 Applicant responded on January 18, 2012, without amending its claims, submitting a declaration that Applicant's inventors had conceived and reduced to practice the application claims prior to the priority date of Lindholm. *Id.*

58. Examiner issued a Final Rejection on March 15, 2012, rejecting Applicant's declaration swearing behind an earlier priority date than Lindholm, finding that the declaration lacked specific facts and did not affirmatively state that the conception and reduction to practice of the application claims was in the United States, a NAFTA country, or a WTO member country. *Id.* Examiner maintained its anticipation rejections under Lindholm. *Id.* Applicant responded on September 17, 2012, canceling application claims 9 and 12-14 without prejudice. *Id.* Applicant further argued that its previous declaration was sufficient to overcome the priority date of Lindholm. *Id.* Applicant also shortened its abstract to overcome Examiner's comments on the specification.

59. Examiner issued its second Non-Final Rejection on December 6, 2012. Ex. 1002. Examiner again rejected Applicant's declaration swearing behind an earlier priority date than the Lindholm reference. *Id.* Examiner again found that the declaration consisted of vague and general statements related to conception and reduction to practice, and Examiner again found that the declaration failed to specify that these events occurred in the United States, a NAFTA country, or a WTO member country. *Id.* Finally, Examiner reiterated its anticipation rejections and found new indefiniteness and obviousness arguments on several claims. *Id.*

60. Applicant responded on June 6, 2013, further reducing its abstract and amending the claims to overcome the indefiniteness rejections. *Id.* For independent application claim 2, Applicant changed "one of the plurality of inputs" to "data from the general purpose register block," and, for dependent application claims 8 and 10, Applicant changed "one of the plurality of inputs" to "data." *Id.* Applicant further argued that the requisite level of detail existed in its declaration, but the detail was in the referenced exhibits rather than the declaration itself. *Id.* Applicant also included additional detail within the declaration exhibits. *Id.* Applicant further argued that Examiner rejected its claims on obviousness grounds based on Examiner's misinterpretation of the cited prior art. *Id.*

61. Examiner issued its second Final Rejection on August 8, 2013. Ex. 1002. The Examiner accepted Applicant's new declaration, but without any

substantive analysis or comments. Examiner then rejected Applicant's claims under U.S. Patent No. 7,646,817 ("Shen") in view of U.S. Patent No. 6, 697,074 ("Parikh"). *Id.*

62. Applicant further initiated an interview on January 29, 2014, to overcome Examiner's final rejections. Ex. 1002. Examiner did recognize that certain application claims would be allowed. *Id.* However, Examiner considered Shen to describe independent claims in the application system because it has a GPU that can perform both pixel and vertex shading. *Id.* Applicant argued that its claims differed from Shen because the application claims had a CPU that can perform dual shading while Shen only disclosed a GPU. *Id.* The parties did not reach an agreement in the interview. *Id.*

63. On February 4, 2014, Applicant filed a Terminal Disclaimer to obviate double-patenting over prior patents. Ex. 1002. Applicant amended its claims to be clear that the method claims were carried out by a unified shader, and the processor was within the unified shader. *Id.* Applicant also combined claims 15 and 16, per Examiner's comments. *Id.*

64. On February 14, 2014, Examiner issued a Notice of Allowance. Ex. 1002.

65. Finally, the '454 patent was challenged in an inter partes review on April 3, 2017. The petitioner withdrew its IPR Petition on September 28, 2017,

before the Board made an institution decision. *Id.* The Board terminated the proceedings the next day. *Id.*

66. I understand that ATI's evidence of prior invention of the "ATI Unified Shader" was subsequently found insufficient to antedate the '685 patent after a full adjudication on the merits. *See Certain Consumer Electronics and Display Devices with Graphics Processing and Graphics Processing Units Therein*, Inv. No. 337-TA-932, Final Initial Determination (October 9, 2015) (Ex. 1015).

VIII. LEVEL OF ORDINARY SKILL IN THE ART

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

IX. GROUND 1: LINDHOLM '685 PATENT AND LINDHOLM '913 PATENT (CLAIMS 1-11)

68. I have reviewed and am familiar with the following prior art references:

- a. U.S. Patent No. 7,038,685 ("Lindholm '685 patent") was filed on June 30, 2003 and issued on May 2, 2006. Ex. 1005.

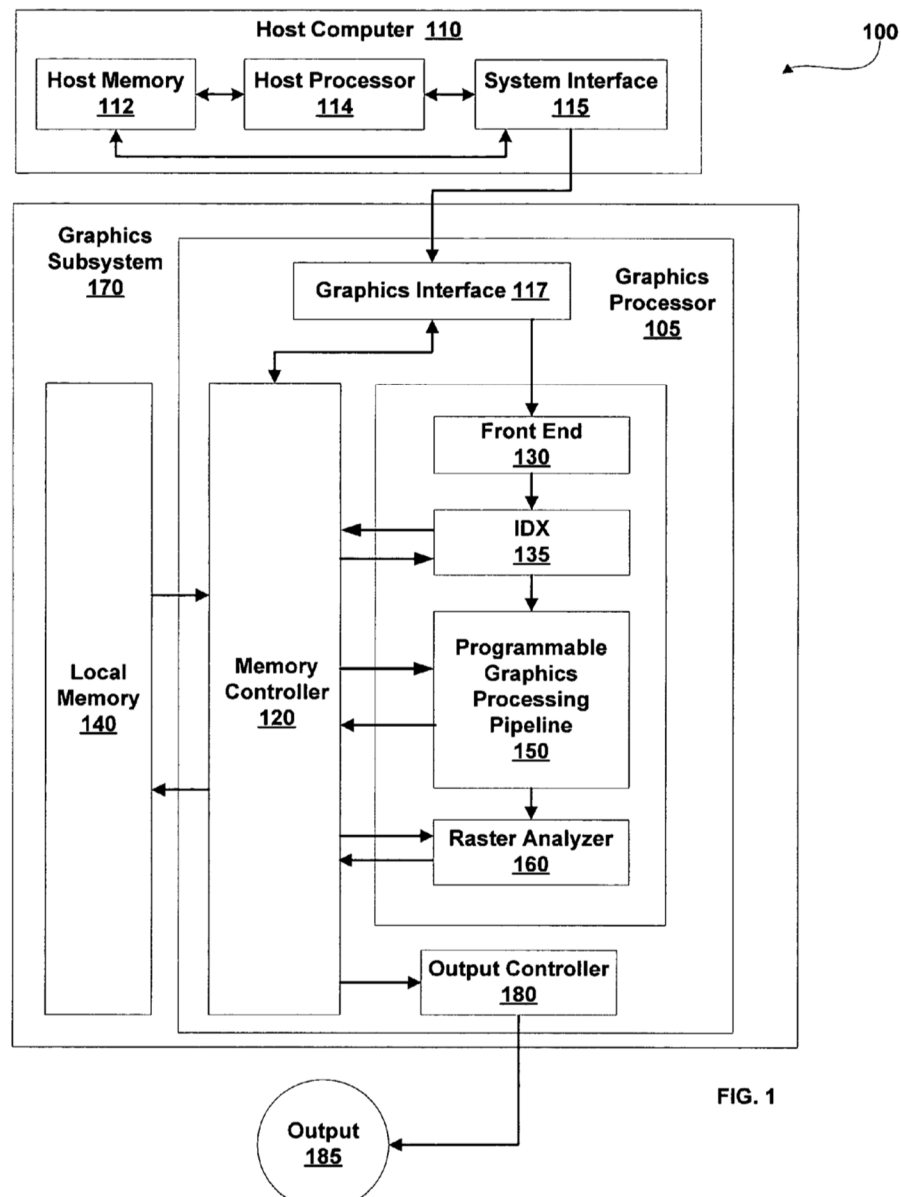
b. U.S. Patent No. 7,015,913 (“Lindholm ’913 patent”) was filed on June 27, 2003 and issued on March 21, 2006. Ex. 1004.

69. It is my opinion that the combination of the Lindholm ’685 Patent and the Lindholm ’913 Patent renders claims 1-11 of the ’454 Patent obvious.

A. The Lindholm ’685 Patent

70. U.S. Patent No. 7,038,685 (“the ’685 patent”), entitled “Programmable Graphics Processor for Multithreaded Execution of Programs,” was filed on June 30, 2003 – less than a week after the ’913 patent – and issued on May 2, 2006.

71. The '685 patent describes a unified shader architecture that addresses the issue of thread assignment for the simultaneous execution of vertex and pixel data. Figure 1 of the patent shows the overall graphics system architecture.



72. The system includes a Host Computer 110 and a Graphics Subsystem 170. The Graphics Processor 105 within the graphics subsystem contains a Programmable Graphics Processing Pipeline 150, shown in greater detail in Fig. 2.

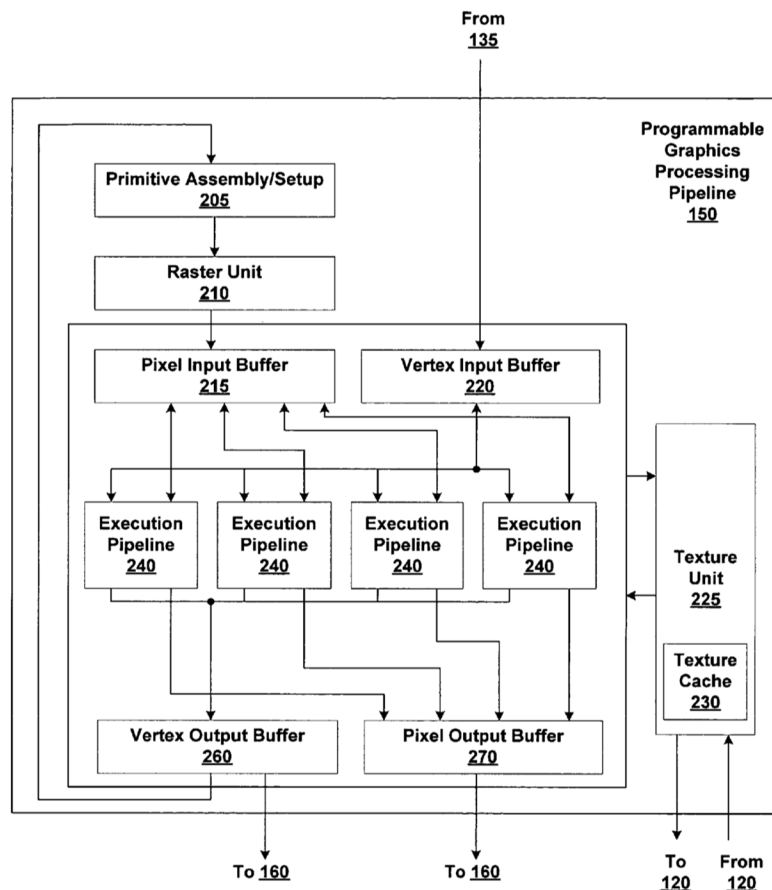


FIG. 2

73. The Vertex Input Buffer 220 stores “surfaces, primitives, or the like” processed by one or more programmable Execution Pipeline 240. Ex. 1004 at 4:21-27. Similarly, the Pixel Input Buffer 215 stores “pixels and fragments” that are also processed by the Execution Pipelines. Ex. 1004 at 5:11-14. Each Execution

Pipeline 240 contains at least one Multithreaded Processing Unit 400. Ex. 1004 at 4:27-29 and Fig. 4.

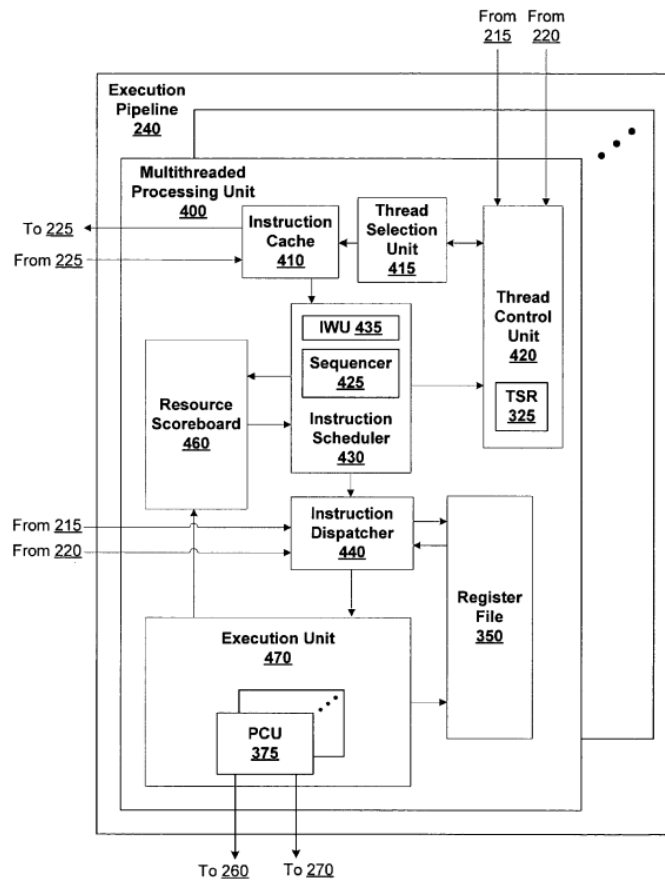


FIG. 4

74. As depicted above, the multithreaded processing unit 400 of the '685 patent comprises various components that enable the unit to carry out multi-threaded execution of program instructions. It can accommodate numerous threads, each processing vertices or pixels. *Id.* at 6:52-59. When a thread is available, each multi-threaded processing unit sends a signal to the Vertex Input Buffer or the Pixel Input Buffer when a vertex or a pixel can be accepted for processing. *Id.* at 4:31-36, 5:23-25.

75. The '685 patent utilizes a sophisticated arbitration mechanism to decide whether vertex or pixel data is to be processed in its unified shader. This mechanism is primarily facilitated by the Thread Control Unit and the Thread Selection Unit. Ex. 1004 at 6:40-7:36. The Thread Control Unit operates based on a thread allocation priority system. *Id.* This priority system decides whether a vertex or a pixel sample will be allocated to an available thread. *Id.* This priority could be fixed, programmable, or dynamic (for example, consistently prioritizing vertex samples or establishing priority based on the number of vertices and pixels awaiting processing). *Id.*

76. After the Thread Control Unit determines whether vertices or pixels will be assigned to available threads, the Thread Selection Unit uses an “execution priority assigned to each thread type.” This execution priority is used to determine the order in which the vertex and pixel data will be output to the unified shader for processing (which includes the Instruction Cache and Execution Unit). *Id.* at 7:56-8:32. As with the thread allocation priority, the execution priority may be fixed, programmable, or dynamic. *Id.* at 7:60-61.

77. The '685 patent also uses a complex arbitration scheme in the Instruction Cache and Instruction Scheduler to select which instructions from the available threads will be processed. These units can select “any instruction within the IWU 435 [to] be executed based on the availability of computing resources in

an Execution Unit 470 and source data stored in Register File 350.” *Id.* at 7:56-64; 8:47-64. Thus, the ’685 patent employs multiple arbitration schemes at the data level, thread level, and individual instruction. This permits the unified shader to simultaneously process multiple data types and switch between processing vertices and pixels effortlessly.

B. The Lindholm ’913 Patent

78. U.S. Patent No. 7,015,913 (“the ’913 patent”), entitled “Method and Apparatus for Multithreaded Processing of Data in a Programmable Graphics Processor” was filed on June 27, 2003, and issued on March 21, 2006. Ex. 1004.

79. Like the ’685 patent, the ’913 patent discloses a unified shader architecture capable of freely switching between instructions and processing those instructions and data simultaneously. This solved a key problem with prior art graphics processors, as described in the ’913 patent’s Background:

Traditionally graphics processing systems were implemented as fixed function computation units and more recently the computation units are programmable to perform a limited set of operations. In either system, the graphics data elements are processed in the order in which they are received by the graphics processing system. Within the graphics processing system, when a resource, e.g., computation unit or data, required to process a graphics data element is unavailable, the processing of the element stalls, i.e., does not proceed, until the

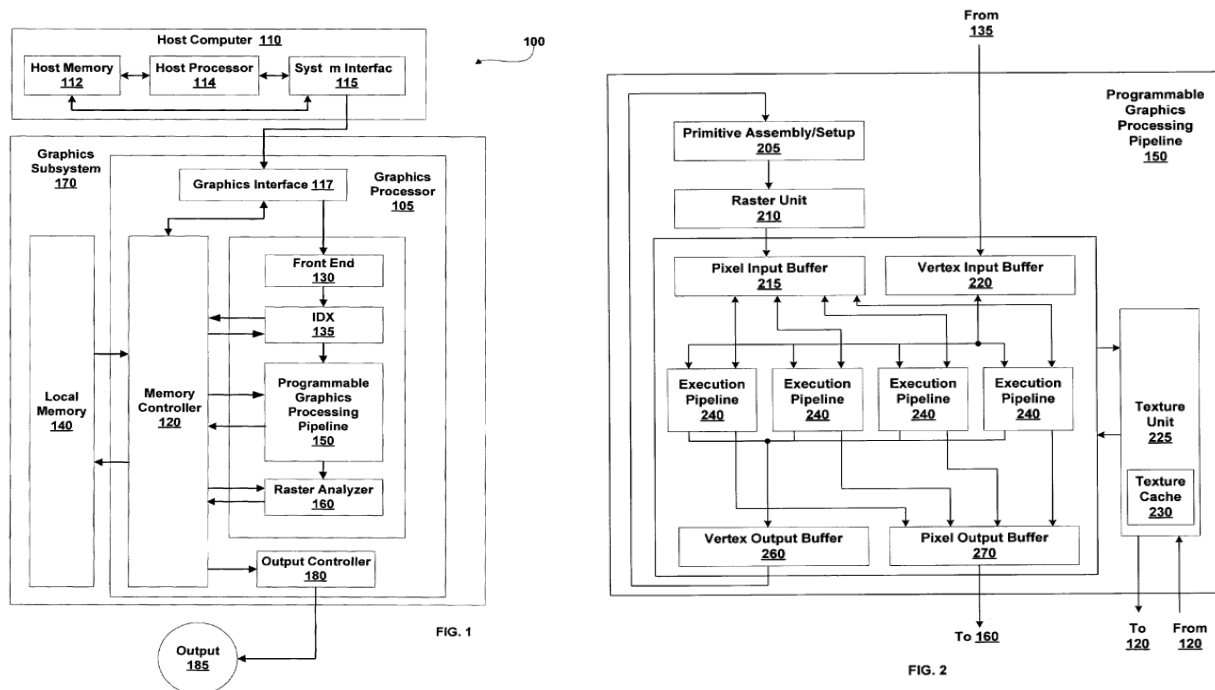
resource becomes available. Because the system is pipelined, the stall propagates back through the pipeline, stalling the processing of later received elements that may not require the resource and reducing the throughput of the system.

Ex. 1004 at 1:27–31.

80. The processor in the ‘913 patent avoided this problem by processing graphics data elements in an order independent from the order in which the pipeline received them. As the ‘913 patent explains:

The graphics processor includes at least one multi-threaded processing unit configured to receive samples in a first order to be processed by program instructions associated with at least one thread. Each multi-threaded processing unit includes a scheduler configured to receive the program instructions, determine availability of source data, and schedule the program instructions for execution in a second order independent of the first order.

Id. at 1:49–57. These elements are illustrated in the figures below:



81. Essentially, the ‘913 patent invented a way to track source data and base program instruction scheduling on the availability of that source data. *Id.* at Abstract. Instead of creating bottlenecks by processing data in the order it was received, the ‘913 patent diverged from the prior art processors, providing a way to execute instructions from another thread when one thread awaited the source data necessary for execution. *Id.* This, in turn, allowed for faster and more efficient graphics processing. *Id.*

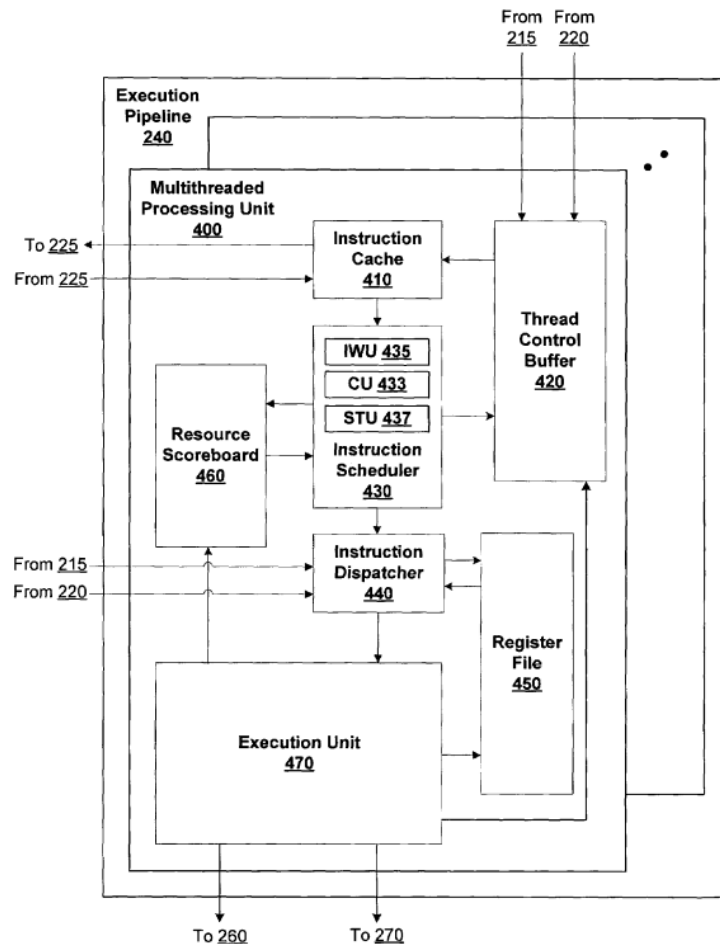


FIG. 4

82. As depicted above, the '913 patent's graphics processor contains many components, which combine to allow the '913 patent's processor to accomplish multi-threaded execution of program instructions in the most efficient order possible.

83. Among these components is a scheduler, which receives program instructions to process the samples. Ex. 1004 at 4:29-32, 5:54-58, 6:63-7:5, 16:48-52, 17:41-44. If a particular program instruction specifies available source data,

the scheduler will schedule that program instruction to be executed because doing so would not stall the system. *Id.* at 8:49-53. If the specified source data is not available, the scheduler will instead schedule a different thread's program instruction for execution. *See id.* In short, the scheduler bases its decision to execute a particular program instruction based on the availability of source data specified by that instruction. *Id.* at 8:49-53; 17:41-44. Program execution is conducted on an instruction-by-instruction basis, and the scheduler switches at will between instructions and threads of various sample types. *Id.* at 17:41-44.

84. A second component, the resource scoreboard, uses the location of each program instruction's required source data to track whether that data is available for processing vertex or pixel threads and whether the necessary computational resources are available. *Id.* at 8:58-60; 9:2-4, 16:53-54, 17:45-46. The scheduler uses this information from the resource scoreboard to determine which program instructions have their source data available and, consequently, which program instructions to schedule for execution. *See, e.g.,* Ex. 1004 at 14:20-30.

85. Once the program instructions have been scheduled for execution, a third component, the instruction dispatcher, gathers the specified source data and scheduled instructions and outputs them to the execution unit to process the

corresponding sample. *Id.* at 9:40-42; 16:55-57; 17:47-50. This completes the processing operation.

1. Combining the Lindholm Patents

86. The '685 and the '913 patents are attributed to the same inventor, John Erik Lindholm, focusing on the same architecture and identical unified shader processing unit. These patents were filed three days apart, with much of their specifications and figures strikingly similar. They are both targeted towards the same multithreaded unified shader. A POSITA at the time of the '454 patent's alleged invention would have been undoubtedly motivated to examine both Lindholm patents and would have identified that they were disclosing different aspects of the same unified shader architecture. In particular, a POSITA would have been motivated to look to the '913 patent for the additional details that it disclosed regarding the multithreaded processing unit of the common unified shader architecture and its scheduling of threads for processing, including the ability to process graphics data elements in an order independent from the order in which the pipeline received them. As described above, the '913 patent discloses additional details regarding the ability of the multithreaded processing unit to track source data and base program instruction scheduling on the availability of that source data. Program execution is conducted on an instruction-by-instruction basis, and the scheduler switches at will between instructions and threads of

various sample types. This improves performance because a stalled thread does not prevent the execution of other threads. Consequently, a POSITA would have been explicitly driven to combine the teachings of the two Lindholm patents, and a POSITA would encounter no difficulty in combining the teachings of the two Lindholm patents.

2. Independent Claim 2

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

2. A unified shader, comprising:

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

[b] a processor unit;

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

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

a. Claim 2 [preamble] – Unified Shader

88. Assuming the preamble is a limitation, each of the Lindholm Patents discloses a unified shader. The complete specifications of both Lindholm '685 and

89. The graphics processor of the '685 patent, Ex. 1005 at Fig. 1, contains a programmable graphics processing pipeline. This pipeline incorporates multiple

execution pipelines, each capable of receiving vertex data from the vertex input buffer 220 and pixel data from the pixel input buffer 215, as shown in Fig. 2.

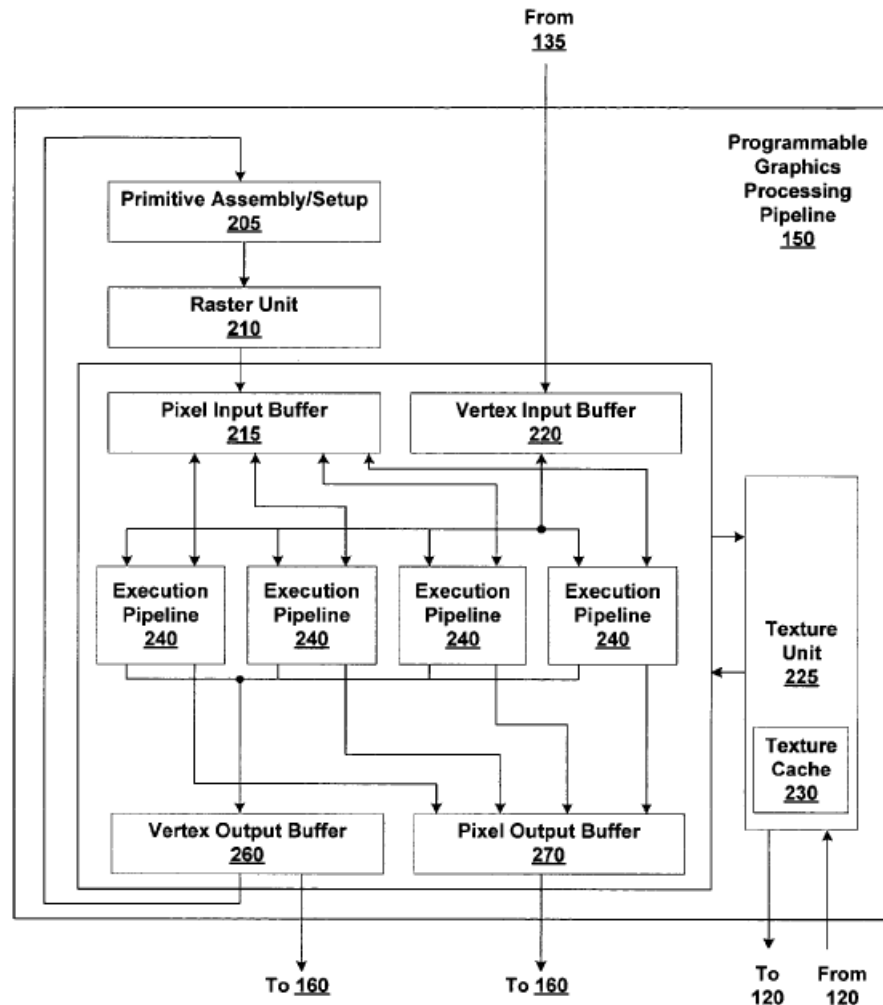


FIG. 2

Ex. 1005 at Fig. 2, 4:22-5:35. *See also* Ex. 1006 at Fig. 2, 4:35-5:45.

“Programmable Graphics Processing Pipeline 150 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample or any other data.” Ex. 1005 at 3:57-64; 4:13-41.

90. Each execution pipeline can perform vertex operations, such as transforming vertices from “object space” to another coordinate system. Ex. 1005 at 4:42-5:10. Each execution pipeline also performs pixel operations, such as “texture mapping, shading, blending, and the like.” *Id.* at 5:11-49.

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

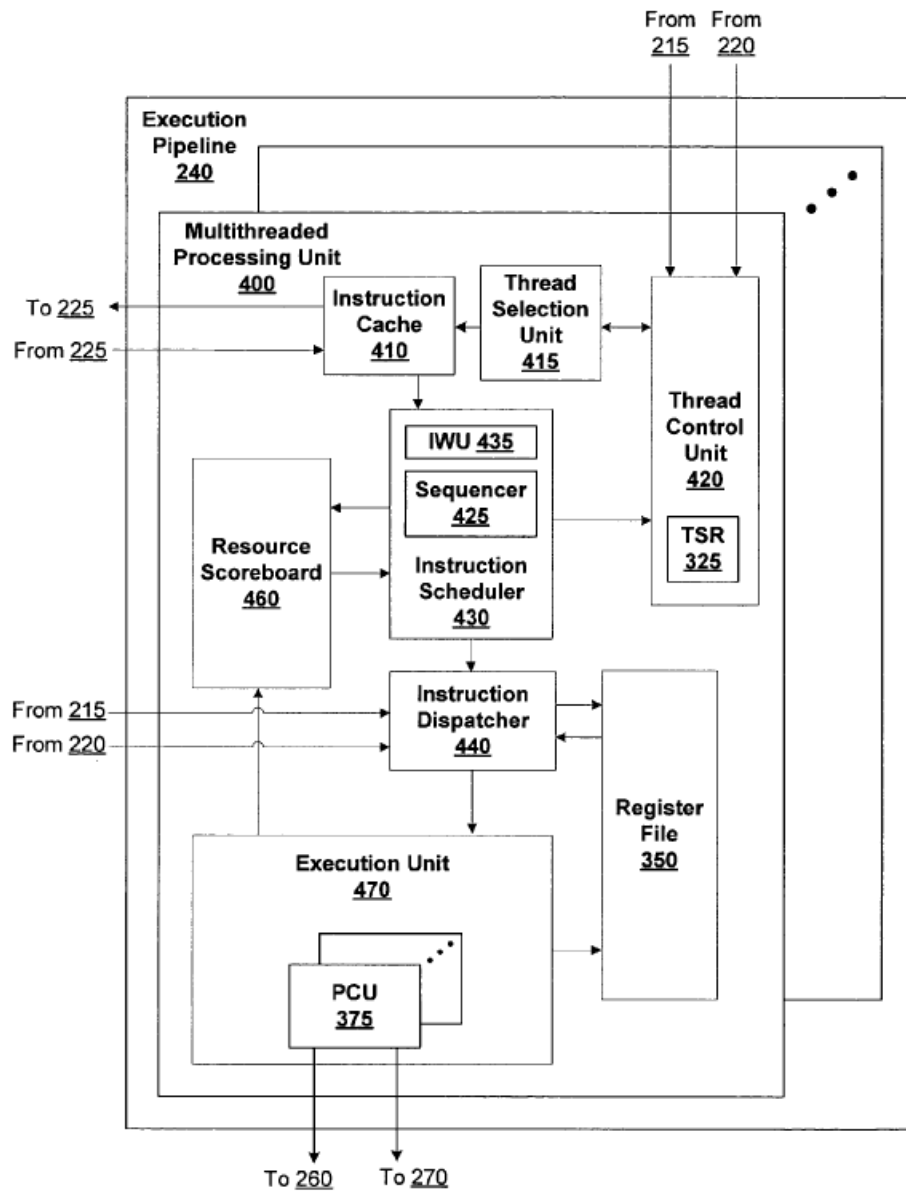


FIG. 4

Ex. 1005 at Fig. 4, 4:27-29; 5:36-49, 8:30-32; Ex. 1006 at Fig. 4, 4:40-42, 8:19-20.

Each multithreaded processing unit can simultaneously process vertex data and pixel data (as described further with respect to claim elements 2[b] and [d]).

FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multithreaded Processing Unit

400. Thread Control Unit 420 includes a TSR 325 to retain thread state data. In one embodiment TSR 325 stores thread state data for each of at least two thread types, where the at least two thread types may include pixel, primitive, and vertex ... thread state data for a thread assigned to a sample may include the sample type, e.g., pixel, vertex ...

Ex. 1005 at 6:38-49. *See also* Ex. 1006 at 7:19-42. The Programmable Graphics Pipeline 150, or alternatively, each Execution Pipeline 240 (and/or components of each of the foregoing) in each of the Lindholm patents, corresponds to the “unified shader” of the ’454 patent.

b. Claim 2[a] – General Purpose Register Block

92. The Lindholm Patents each disclose “a general purpose register block for maintaining data,” which includes both vertex data and pixel data.

Thread Control Unit **320** receives samples ... stored in Pixel Input Buffer **215** and Vertex Input Buffer **220**. Thread Control Unit **320** receives a pointer to a program to process one or more samples. Thread Control Unit **320** assigns a thread to each sample to be processed. ... An instruction specifies the location of source data needed to execute the instruction. Source data, such as intermediate data generated during processing of the sample, is stored in a Register File **350**. In addition to Register File **350**, other source data may be stored in Pixel Input Buffer **215** or Vertex Input Buffer **220**. In an alternate embodiment source data is stored in Local Memory **140**, locations in Host Memory **112**, and the like.

Ex. 1005 at 5:50-6:3. *See also id.* at 7:37-43, 8:47-58.

93. Accordingly, the Register File 350 maintains source data, including vertex data and pixel data. “In addition to Register File 350, other locations of source data include Pixel Input Buffer 215, Vertex Input Buffer 220, locations in Local Memory 140, locations in Host Memory 112, and the like.” Ex. 1005 at 8:54-58. The Thread Control Unit 420 receives vertex data from the Vertex Input Buffer and pixel data from the Pixel Input Buffer. *Id.* at 5:50-6:3.

94. The Pixel Input Buffer and the Vertex Input Buffer also include “a register file, FIFO ... cache, or the like.” Ex. 1005 at 4:22-26, 5:32-35. A POSITA would understand these to be portions of a general-purpose register block. Accordingly, each component contains registers that can maintain vertex and pixel data and individually or collectively constitute “a general purpose register block” as required by claims 1 and 2.

c. Claim 2[b] and [d] – Processor Unit

95. Each Lindholm Patent discloses claim elements 2[b] and [d]. Within each multithreaded processing unit, “Execution Unit 470 can simultaneously process samples of different types [*e.g.*, vertices and pixels], and likewise, execute threads of different types.” Ex. 1005 at 9:47-49. *See also id.* at claims 1, 13, 23, 25, 36, 41, 43 (claiming a multithreaded processing unit that simultaneously processes at least two sample types, such as vertices and pixels). “Execution Unit

470 is configured by the program instruction to simultaneously process samples using PCUs 375 ... and output the processed sample to ... Vertex Output Buffer 260, Pixel Output Buffer 270, or Register File 350.” *Id.* at 9:39-45. The Execution Units and/or the programmable computation units (PCUs) that comprise each Execution Unit satisfy the “processor unit” limitations of claim 2, as well as the processor unit limitations in claims 3-11 and the “processor” of claim 1.

96. Each Execution Unit 470 and its PCUs 375 execute instructions that perform pixel operations (e.g., to generate a pixel color) and vertex operations (e.g., to generate vertex position and appearance data). *See id.* at 4:22-5:35 (describing the processing of vertices and pixels by the PCUs in the Execution Pipelines), 9:33-49, 11:1-14 (shader programs and source data dispatched to the PCUs/Execution Units for execution), 15:27-35. Lindholm’s “samples” include both pixel and vertex operations. *Id.* at 3:59-64. When the “Thread Control Unit 320 gathers source data specified by the program instructions and dispatches the source data and program instructions to at least one PCU 375,” the PCU, in response to receiving the source data and program instructions, “performs computations specified by the program instructions and outputs data” to the Pixel Output Buffer or Vertex Output Buffer. Ex. 1005 at 5:36-49. *See also id.* at 9:33-49.

97. Based on the disclosures of each of the Lindholm Patents, a POSITA would have recognized that the Execution Units/PCUs “execute instructions that generate a pixel color in response to selected data.” For example, the ’685 patent describes how “when a sample can be accepted” from the Pixel Input Buffer 240, “[p]rogram instructions configure programmable computation units (PCUs) within an Execution Pipeline 240 to perform operations [on the pixel data] such as tessellation, perspective correction, texture mapping, shading, blending, and the like,” after which the “processed samples” are output to the Pixel Output Buffer. Ex. 1005 at 5:23-35. A POSITA would recognize that such instructions, when executed by the Execution Units/PCUs, generate a pixel color due to the various pixel operations and that the instructions are executed in response to the Execution Units/PCUs accepting or receiving pixel data. *See also* ’454 Patent at 1:38-48 and 2:8-22 (explaining how the pixel shading part of graphics processing provides the color value for each pixel of a rendered object).

98. Similarly, a POSITA would have recognized that the Execution Units/PCUs “execute instructions that generate vertex position and appearance data in response to selected data.” It was common knowledge for a POSITA that operations conducted on vertices influenced their location and appearance. For instance, the ’685 patent reveals that the execution pipelines, when processing vertex samples, would “transform the second samples from an object-based

coordinate representation (object space) to an alternatively based coordinate system.” This transformation generates vertex position data and applies texture maps, thereby producing appearance data. Ex. 1005 at 4:42-64, 9:33-49.

99. The Execution Units 470 and PCUs 375 are coupled to the Register File 350, as well as to the additional memory locations mentioned in the Lindholm Patents, under claim element 2[a]. These locations constitute the general-purpose register block from where the Execution Units and PCUs obtain vertex and pixel data. See Ex. 1005, FIGS. 1, 4.

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

d. Claim 2[c] - Sequencer

101. The Lindholm Patents each disclose the claimed “sequencer,” which aligns with the combination of the Instruction Cache 410, the Instruction Scheduler 430 (inclusive of the Sequencer 425 and IWU 435), and the Instruction Dispatcher 440. These components work in concert to sustain instructions that instruct the Execution Unit/PCUs to carry out vertex calculation and pixel calculation

operations on chosen data housed in the general-purpose register block, as previously elaborated in reference to claim 2[d].

102. The Instruction Cache 410 outputs program instructions for threads to the Instruction Scheduler 430. Ex. 1005 at 8:33-46. “Instruction Scheduler 430 contains storage resources to store a predetermined number of instructions” and evaluates whether instructions can be executed by the Execution Unit/PCUs depending on the availability of computation resources and source data in Register File 350. *Id.* at 8:47-9:4. “When Instruction Scheduler 430 determines which instructions and associated threads will be executed,” it processes instructions using Sequencer 425. *Id.* at 9:5-32. “Instruction Dispatcher 440 [then] gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 specified in an instruction and outputs the instruction and source data to Execution Unit 470 including at least one PCU 375.” *Id.* at 9:33-49. Accordingly, the “sequencer” is coupled to the “processor unit” (Execution Unit/PCUs) and to the general-purpose register block (Register File 350 and the Vertex/Pixel Input Buffers). *See id.* Fig. 4.

3. Independent Claim 3

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

3. A unified shader comprising:

- [a] a processor unit operative to perform vertex calculation operations and pixel calculation operations; and
- [b] shared resources, operatively coupled to the processor unit;
- [c] the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.

a. Claim 3 [preamble] – Unified Shader

104. See *supra* at §2[preamble].

b. Claim 3[a] and [c] – Processor Unit

105. Each of the Lindholm patents discloses Execution Units/PCUs as the claimed “processor unit,” performing both vertex and pixel calculation operations. See *supra* at §2[b] and [d].

106. Each of the Lindholm patents discloses that the Execution Units/PCUs are operative to perform either vertex or pixel calculation operations based on a priority, which may be a thread allocation priority or a thread execution priority, and which may be fixed, programmable, or dynamic. The priority may be fixed, always giving priority to vertex processing (claim 4) or pixel processing (claim 5), or it may be dynamic, based on the vertex or pixel workload (claim 11) or the number of resources available, such as “space available in the store,” *i.e.*, the Register File 350 and Vertex/Pixel Input Buffers (claim 5).

107. Since vertex and pixel operations occur simultaneously within each multithreaded processing unit, data is allocated to accessible threads based on a “thread allocation priority.” This priority is determined for each sample type (vertex or pixel) and “may be fixed, programmable, or dynamic.” Ex. 1005 at 6:60-7:36 and Ex. 1006 at 7:43-58. Vertex or pixel threads are subsequently chosen for processing by the Thread Selection Unit using an “execution priority.” This priority is set for each sample type assigned to the threads (*e.g.*, vertex and pixel) and “may be fixed, programmable or dynamic.” Ex. 1005 at 7:56-6

108. With a fixed vertex thread allocation priority, vertex calculation operations would be performed until enough “shared resources” become available, and then pixel calculation operations would be performed. “In one embodiment the thread allocation priority may be fixed, always giving priority to allocating vertex threads and pixel threads are only allocated if vertex samples are not available for assignment to a thread.” Ex. 1005 at 6:60-7:5. Conversely, a fixed pixel thread allocation priority would prioritize processing pixel threads, and vertex operations would only be performed if no pixel samples were available for thread assignment. *Id. See also id.* at 10:17-54 (describing Fig. 5B, a flow diagram showing the application of thread allocation priority), 16:10-55 (describing Fig. 9B, a flow diagram showing the application of thread execution priority).

109. In alternate scenarios, 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. If, for instance, a dynamic thread allocation priority is configured to favor pixel operations, then priority will be assigned to pixel operations according to the number of pixel samples queued for processing. “Dynamically modifying a thread allocation priority for pixel samples based on the amount of sample data in Pixel Input Buffer 215 permits Pixel Input Buffer 215 to drain faster.” *Id.* at 7:6-36. “For example, a dynamically determined thread allocation priority may be determined based on a number of ‘pending’ pixels.” *Id.* On the other hand, a dynamic thread allocation priority that gives precedence to vertices will prioritize vertex operations considering the number of vertices in line for processing. *Id.*

110. Alternatively, the thread allocation priority may be completely dynamic, prioritizing either vertex or pixel workload, depending on which is greater. “In another embodiment, Thread Selection Unit 415 is configured to read thread entries based on the amount of sample data in Pixel Input Buffer 215 and the amount of sample data in Vertex Input Buffer 220.” *Id.* at 7:65-8:14. Likewise, “a dynamically determined thread execution priority” can be determined based on the number of pixels or the number of vertices that are waiting to be processed. *Id.* *See also id.* at 15:42-57 (thread execution priority determined for each thread type,

pixel or vertex, “based on the amount of sample data in Pixel Input Buffer 215 and the amount of sample data in Vertex Input Buffer 220”).

111. In addition to giving a fixed priority to vertex or pixel operations, the priority can be set or changed based on the availability of resources, including space available in the “general purpose register block” or “store.” “Each clock cycle, Instruction Scheduler 430 evaluates whether *any* instruction within the IWU 435 can be executed based on the availability of computation resources in an Execution Unit 470 and source data stored in Register File 350,” which includes data from either the “Pixel Input Buffer 215 [or] Vertex Input Buffer 220.” Ex. 1005 at 7:56-64; 8:47-64; 9:2-4; 9:50-62; Ex. 1006 at 8:47-64. Thread allocation or execution decisions can also be based on space available in Register File 350, or “the amount of storage for intermediate data generated during processing of a sample,” and the number of threads of each sample type that can be executed in a particular embodiment. *Id.*, and at 7:37-40, 12:9-36, 16:19-24.

112. Because each multithreaded processing unit is multi-threaded, and multiple multi-threaded processing units can exist across multiple execution pipelines, the Lindholm patents contemplate many simultaneously executing threads. Ex. 1005 at 12:9-14. “Execution Unit 470 can simultaneously process samples of different types, and, likewise, execute threads of different types.” *Id.* at

9:33-49. “Likewise, a number of threads of each sample type that may be executed simultaneously may be limited in each embodiment.” *Id.* at 12:14-16.

113. As vertex and pixel operations occur simultaneously, the patents further explain how the distribution of processing resources between vertex and pixel operations can be altered during operation. *Id.* at 12:9-27. *See also* Ex. 1006 at 7:6-12. Consequently, the Execution Units/PCUs can conduct vertex operations and pixel operations at varying stages of completion by transitioning between instructions in the Instruction Cache and Instruction Scheduler.

114. Even though none of the claims in the '454 patent mandate a multi-threaded processor or a switch between “unfinished” pixel threads and “unfinished” vertex threads, these concepts are also disclosed in the Lindholm Patents. This illustrates one method through which the Lindholm Patents describe the ability to freely alternate between vertex and pixel operations at various stages of completion. *See generally* Ex. 1006 at 7:19-10:3; 13:17-14:62.

115. The Lindholm '913 patent elaborately presents many algorithms, conditions, and hardware mechanisms to determine which instructions to execute. These instructions are chosen from any hardware thread context within the multithreaded processing unit.

116. The Lindholm '913 patent outlines how instructions from diverse threads stored in the Thread Control Buffer are selected for execution. This process

of scheduling and executing instructions at the instruction level commences with the Instruction Scheduler, which can output multiple instructions per clock cycle, corresponding to multiple threads. Ex. 1006 at 8:21-46.

When Instruction Scheduler 430 determines which instructions and associated threads will be executed, Instruction Scheduler 430 outputs at least one instruction to a dispatcher, Instruction Dispatcher 440, updates destination register status and computation resource availability in Resource Scoreboard 460 and increments each program counter associated with the threads in Thread Control Buffer 420 associated with the at least one instruction output.

Ex. 1006 at 9:24-30.

117. The Lindholm '913 patent further illustrates how the execution units can concurrently process multiple samples. This is achieved through the Instruction Scheduler and Instruction Dispatcher issuing instructions originating from multiple threads (*i.e.*, freely alternating between instructions of varying threads and thread types) that are stored in the multithreaded processing unit:

Rather than processing one sample as shown in FIG. 5, Multithreaded Processing Unit 400 receives a stream of samples, additional threads are assigned to each sample and instructions are fetched for each thread. ***Instruction Scheduler 430 determines which instructions can be scheduled, choosing amongst instructions that process different samples. In this manner Multithreaded Processing Unit 400 can simultaneously process one or more samples using at least one***

program, where each sample may be processed in an order that is independent of the order in which the samples were received by Multithreaded Processing Unit 400. Likewise, each Multithreaded Processing Unit 400 can simultaneously process one or more samples using at least one program, where each sample may be processed in an order that is independent of the order in which the samples were received by Execution Pipeline 240.

Ex. 1006 at 13:24-39.

118. In this manner, the Lindholm '913 patent can switch back and forth between issuing vertex operations and pixel operations to the execution unit. Each execution unit may have multiple threads functioning concurrently (at least at different stages of its pipeline). Furthermore, it can toggle between vertex and pixel threads to guarantee dynamic load balancing in the graphics processor.

119. As described above, a POSITA would recognize that each of the Lindholm Patents discloses and describes the same multithreaded processing unit with the same execution unit. The additional details provided in the Lindholm '685 patent would be easy and straightforward to apply to the disclosed embodiment in the Lindholm '913 patent and vice versa.

c. Claim 3[b] – Shared Resources

120. I incorporate by reference the discussion *supra* at §2[a]. The “shared resources”¹ include the “general purpose register block,” (’454 patent at 5:32-52), which in the ’685 patent includes Register File 350, as well as the Pixel Input Buffer 215 and Vertex Input Buffer 220. *Id.* Each are “coupled to” the Execution Unit/PCUs through Instruction Dispatcher 440, which gathers vertex and pixel data from those memory locations and outputs it to the Execution Units/PCUs, and thus the Execution Units/PCUs are “operative to use the shared resources for either vertex data or pixel information.” *Id.*; Ex. 1005 at Fig. 4, 9:33-49.

4. Independent Claim 4

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

- 4. A unified shader comprising:
 - [a] a processor unit operative to perform vertex calculation operations and pixel calculation operations; and
 - [b] shared resources, operatively coupled to the processor unit;
 - [c] the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.

a. Claim 4 [preamble] – Unified Shader

122. See *supra* at §2[preamble].

¹ “Shared resources” are understood to refer to components that store both vertex and pixel data.

b. Claim 4[a] and [c] – Processor Unit

123. *See supra* at §3[a] and [c].

c. Claim 4[b] – Shared Resources

124. *See supra* at §3[b].

5. Independent Claim 5

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

4. A unified shader comprising:

[a] a processor unit;

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

a. Claim 5 [preamble] – Unified Shader

126. *See supra* at §2[preamble].

b. Claim 5[a] – Processor Unit

127. *See supra* at §2[b].

c. Claim 5[b] – Sequencer

128. *See supra* at §§2[c], 3[a] and [c].

6. Dependent Claim 6

129. Dependent claim 6 recites, “The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.” As explained *supra* at §2[c], the “sequencer” in the ’685 patent includes Instruction

Dispatcher 440, circuitry operative to fetch data from memory. “Instruction Dispatcher 440 gathers the source data from Pixel Input Buffer 215, Vertex Input Buffer 220 or Register File 350 ... and outputs the instruction and source data to Execution Unit 470.” Ex. 1005 at 9:33-49, 11:9-14, 15:24-30.

7. Dependent Claims 7 and 10

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

131. The '685 patent reveals the Thread Selection Unit 415 as the claimed “selection circuit” and the Thread Control Unit 420 as the claimed “arbiter.” The Thread Selection Unit operates in response to control signals delivered by the Thread Control Unit 420. It reads thread entries (including thread state data) from the Thread Control Unit, which may allocate threads or samples to threads based on the thread allocation priority or execution priority. *Id.*; Ex. 1005 at 6:60-7:5, 7:6-8:17; 15:14-17:7. Moreover, the “Thread Selection Unit 415 is configured to read thread entries based on the thread execution priority assigned to each thread type” or “based on the amount of sample data in Pixel Input Buffer 215 and the amount of sample data in Vertex Input Buffer 220,” which signals are provided by the Thread Control Unit. *Id.* and at 7:56-8:14, 16:10-55, Fig. 3-4 (thread execution

priority register shown as part of Thread Control Unit). To the extent not expressly disclosed, it would have been obvious to a POSITA to provide the priority control signal from the Thread Control Unit to the Thread Selection Unit.

132. The Thread Selection Unit outputs thread entries and other thread data, such as state data, which provides information to the “store” (Register File 350) at least through the threads selected for execution, using the Instruction Cache, Instruction Scheduler, and Instruction Dispatcher, as well as the Execution Unit. Ex. 1005 at 8:15-9:67. Moreover, information (including intermediate data) is provided to Register File 350 due to the threads selected for execution. *Id.* at 5:50-6:3, 7:37-40, 8:47-9:4, 9:33-66.

8. Dependent Claim 8

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

9. Dependent Claim 9

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

10. Independent Claim 11

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

11. A unified shader comprising:

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

[b] an instruction store and

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

a. Claim 11 [preamble] – Unified Shader

136. *See supra* at §2[preamble].

b. Claim 11[a] and [c] – Processor Unit

137. *See supra* at §§2[c], [d], 3[a], [c].

c. Claim 11[b] – Instruction Store

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

11. Independent Claim 1

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

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

[a] performing vertex manipulation operations and pixel manipulation operations by transmitting vertex data to a general purpose register block, and performing vertex operations on the vertex data by a processor within the unified shader unless the general purpose register block

does not have enough available space therein to store incoming vertex data; and

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

140. Claim 1 is a method claim that merely puts the limitations of apparatus claim 3 into method claim format. For example, the steps to be performed in claim 1 align with the functional elements of the processor unit, and the availability of space in the general-purpose register block in claim 1 (which stores both vertex and pixel data) aligns with the availability of shared resources in apparatus claim 3. As discussed earlier under claim 2, the Execution Units 470 and PCUs 375 receive vertex data and pixel data from the general-purpose register block, which includes the Register File 350, the Vertex Input Buffer 220 and the Pixel Input Buffer 215. Similarly, the Thread Control Unit 420 receives vertex data from the Vertex Input Buffer and pixel data from the Pixel Input Buffer. Ex. 1005 at 5:50-6:3. “Execution Unit 470 is configured by the program instruction to simultaneously process samples using PCUs 375 ... and output the processed sample to ... Vertex Output Buffer 260, Pixel Output Buffer 270, or Register File 350.” *Id.* at 9:39-45. A POSITA would understand the Vertex Output Buffer and

Pixel Output Buffer to also be portions of a general-purpose register block. A POSITA would understand that the Thread Control Unit could not send data to the Execution Units or PCUs if the data could not be stored in the general-purpose register block. Therefore, the data processing cannot proceed without sufficient storage in the general-purpose registers. As discussed in connection with claim limitations 3[a] and [c], thread allocation priority and thread execution priority can favor vertex operations or pixel operations and can be changed dynamically based on the availability of resources, including space available in the general-purpose register block, such that vertex operations could be given priority as long as there are sufficient shared resources (including space in the general-purpose register block) for vertex processing, and when there is not, prioritizing pixel operations. Accordingly, the limitations of claim 1 are at least obvious based on the '685 and '913 patents and are met for the same reasons as claims 2 and 3.

X. GROUND #2: CLAIMS 1-11 BASED ON THE COMBINATION OF AMANATIDES AND KOHN

141. I have reviewed and am familiar with the following prior art references:

- a. John Amanatides and Edward Szurkowski, A simple, flexible, parallel graphics architecture, In Proceedings of Graphics Interface at 155-160 (Canadian Information Processing Society

1993) (“Amanatides”) was published in Proc. Graphics Interface ’93 in May 1993. Ex. 1006.

- b. Les Kohn and Neal Margulis, Introducing the Intel i860 64-bit Microprocessor, IEEE, Volume 9, Issue 4, pages 15-30, August 1989. Ex. 1007.

142. It is my opinion that the combination of Amanatides and Kohn renders claims 1-11 of the ’454 Patent obvious.

143. The ’454 patent neither discloses nor asserts a multi-threaded processor. Instead, it claims, at best, a pipelined processor capable of concurrently executing multiple instructions relevant to different data types, such as vertex and pixel data. It describes the operation of the processor unit. Vertex and pixel data have the potential to be processed simultaneously, thanks to separate partitions in the processor for floating-point calculations (required for vertex operations) and scalar/integer calculations (required for pixel operations). Ex. 1001 at 3:4-10; 4:40-51. This concept was not groundbreaking, and in fact, it was invented ten years earlier by Amanatides, who employed the Intel i860 processor to develop a unified shader architecture that could dynamically load balance based on the processor's workload. The i860 processor used by Amanatides had multiple partitions, including one for floating-point calculations (for vertex operations) and another for integer calculations (for pixel operations).

A. Amanatides

144. The paper by Amanatides, titled “A Simple, Flexible, Parallel Graphics Architecture,” presents a unified shader architecture incorporated into a parallel graphics processing system. Amanatides conceived this system to be adaptable, cost-effective, and adequately powered to manage traditional interactive graphics in addition to more advanced graphics applications. Ex. 1007 at Abstract. A decade later, analogous driving factors resulted in the submission of the ‘454 patent.

145. In creating his system, Amanatides “wanted an architecture that minimized graphics hardware. The result, we believe, is an inexpensive, powerful and flexible system The stress was on low cost, flexibility and power. Like the PXM we wanted the new machine to be flexible; we wanted something to perform both interactive graphics and ray tracing fast.” *Id.*

146. The graphics processing framework developed by Amanatides employs DRAM to negate the requirement for SRAM, thereby enabling a greater memory placement per unit area. Ex. 1007 at 158. Furthermore, it incorporates Intel i860 processors, specifically optimized for graphics, with a powerful floating-point unit and on-chip caches. These attributes help to counteract the lag associated with DRAM. *Id.*

147. The system developed by Amanatides leverages priority settings and achieves automatic load-balancing of graphics processing tasks for both vertex and pixel data—a technique that offers a host of benefits:

First, the design is auto load balancing. For example, if there is a lot of texture mapping, the processors spend most of their time on this. If most of the primitives are simple, then the processors spend the majority of their time on the front end computation. Alternately, if the computation is ray tracing, no front-end processors are wasted idling. Second, the approach tends to be simpler, requiring powerful general-purpose processors instead of special-purpose graphics hardware. Because VLSI technology is making these single-chip microprocessors increasingly available this results in a smaller, simpler board layout. Finally, it is a more flexible approach. Since the hardware pipeline is not optimized for a particular algorithm, there is a lot of freedom to change or enhance capabilities. For example, higher-order primitives can be used (NURBS). If necessary, the model can be distributed amongst the processors (assuming that they have the memory) without introducing inefficient asymmetries.

Ex. 1007 at 157.

148. Amanatides' system employs a single processing unit capable of performing all the required front-end operations, including both vertex (G) and pixel (R) processing:

[T]he front and back ends are collapsed onto the same processors. In this case, the host distributes primitives in a round-robin manner to each of the processors. Each processor performs the frontend 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. Because, the primitives are distributed in this manner, each processor does the same back-end operations as in previous designs but only $1/n$ th of the front-end operations (assuming there are n processors).

Ex. 1007 at 157.

149. Amanatides illustrates this system in the diagram below, identifying the unified shader unit as the “G/R” unit.

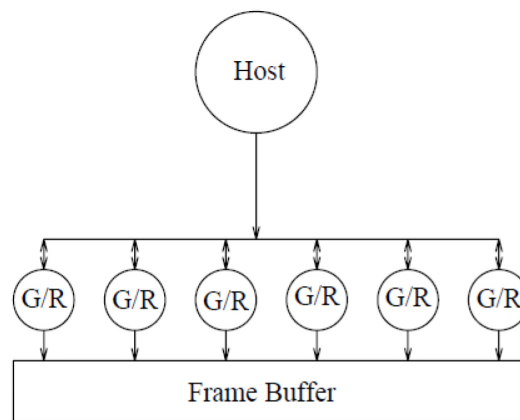


Figure 3

150. This combined unit overcomes the inefficiencies of using dedicated vertex and pixel processors. Previously, “[s]can converting small polygons meant that each processor only drew a few pixels per polygon. However, every processor has to do all the edge setup calculations, and amortizing this over only a few pixels is expensive. When performing interactive graphics the bottlenecks in the system would change depending on the types of rendering involved. If there was texture

mapping, the pixel nodes were the bottleneck. If only simple shading was performed, the pipe nodes were the bottleneck.” Ex. 1007 at 157.

151. By creating a processor that can switch between vertex and pixel processing, Amanatides enabled his system to engage in automatic load balancing.

Consider the following scenario: The host is so fast that it fills all the input FIFOs of all the processors with GM packets. Each processor converts the GM into a RM and puts it in its output FIFO. Their output FIFOs start to fill. What happens if its output FIFO becomes full? If the input FIFO has a RM it is consumed (good). However, if the next packet is a GM it must eventually be sent out and the processor blocks because there is no more room in its output FIFO. If something like this happens at several processors deadlock occurs.

The best way to handle deadlock is to design it out in the first place. In our design we can guarantee that no deadlock will occur if the output FIFO at each processor is not full. We have to make sure that this never happens. RM packets are consumed; they will not cause trouble. It is the GM packets that must be taken care of.

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

Ex. 1007 at 158-59.

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

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

Ex. 1007 at 157.

153. Additionally, Amanatides had the ability to seamlessly toggle between vertex and pixel processing within the same processor unit. This was done with the objective of achieving automatic load balancing between vertex and pixel processing within his graphics system.:

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

...

There are two types of packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. As well, there is a fair scheduling policy in which a second packet from any FIFO cannot get onto the bus until all other FIFOs are first given a chance to send theirs.

...

As was mentioned earlier packets on the bus have two priorities, low and high. Low priority packets cannot get onto the bus if high priority

packets are waiting. If we map GMs to low priority packets and RMs to high priority packets then the solution is in sight.

Ex. 1007 at 158-59.

1. Kohn

154. In a 1989 Feature article for the Institute of Electrical and Electronics Engineers (IEEE), Kohn describes the Intel i860 processor (“Intel i860”) – a 64-bit, RISC-based microprocessor capable of executing parallel instructions. Ex. 1007 at 15. Designed to “incorporate a 3D graphics unit that supports interactive visualization of results,” the Intel i860 was divided evenly into “blocks for integer operations, floating-point operations, and instruction and data cache memories.” *Id.* This design feature allowed Intel to avoid off-chip delays, thus enabling users to push the processor beyond the conventional speeds of 33- and 40-MHz. *Id.*

155. The architecture of Intel’s i860 “provide[d] the data and instruction bandwidth needed to support multiple operations in each clock cycle.” *Id.* Moreover, by balancing performance between integer and floating-point computations, the Intel i860 “produce[d] the raw computing power required to support demanding applications such as modeling and simulations.” *Id.* Essentially, through parallel execution, Intel’s i860 could process graphics faster and more efficiently. *Id.*

156. This enhanced processing capability proved especially beneficial for complex operations such as 3D image processing, which “requires the use of integer operations for shading and hidden surface removal.” *Id.* at 24. Owing to its capacity to handle vertex data (in the floating-point units) and pixel data (in the integer/scalar units) simultaneously, the i860's graphics unit could leverage the floating-point registers and operate in parallel with the core. *Id.* This ability facilitated efficient execution of complex transformations and sophisticated lighting calculations necessary for producing 3D graphics, as well as carrying out other complex tasks. *Id.*

157. The processor could “support multiple operations in each clock cycle,” facilitating “balanced performance between integer and floating-point computations.” *Id.* The concurrent execution of these data types (floating point and scalar/integer) accelerated graphics processing. *Id.* at 16-17. This precise functionality is how the ‘454 patent characterizes the processor unit in its “unified shader.” Ex. 1001 at 3:4-17, 4:40-51.

158. Given its design purpose for graphics processing, the Intel i860 processor could effectively manage vertex and pixel data. For instance, the “floating point hardware of the CPU efficiently performs the transformation calculation and advanced lighting calculations required for 3D graphics.” Ex. 1007 at 16 (describing vertex operations). Moreover, the “use of integer operations for

shading and hidden-surface removal” was a prerequisite for the display of 3D images. *Id.* (describing pixel operations). Therefore, the Intel i860 could operate on vertex data (in the floating point units) concurrently with pixel data (in the integer/scalar units).

B. The '454 Patent Does Not Claim a Multithreaded Graphics Processor

159. The '454 patent claims the ability of the unified shader to simultaneously perform vertex manipulation operations and pixel manipulation operations. But the '454 patent does not claim, nor does the specification describe a *multi-threaded* unified shader. The word “thread” does not even appear in the '454 patent. A POSITA would understand the '454 patent to claim and describe a pipelined single-core, single-threaded processor. That processor architecture would allow for the simultaneous processing of multiple data types, as I described above in the background section.

160. The '454 patent states at column 5, lines 31-35, that “The unified shader 62 has the 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.” This is the only mention of the term “programs,” and the patent does not further specify its technical meaning. The

technical meaning of the term “program” is not synonymous with the term “thread,” and “freely switching between programs” is not synonymous with “multi-threading.”

161. A program is a sequence of instructions that run in the same memory context, whereas each thread can have its own thread context that is not necessarily the same as the context of another thread. Programs can consist of multiple threads, but not the other way around. The patent neither specifies nor requires that the unified shader be capable of running independent programs. It also does not mention the word thread, multi-threading, thread context, or context switch. The description of the unified shader does not specify how it would be able to “freely switch between such programs” or how it would achieve multi-threading with context switching.

162. The claims of the ‘454 Patent are satisfied as long as the unified shader can process vertex and pixel instructions simultaneously, which can be achieved with a single-core, single-threaded pipelined processor.

163. I have read Patent Owner’s Responses in several other *Inter Partes* Review proceedings involving the ’454 patent and its parent patents. In these filings, Patent Owner discusses multi-threaded unified shader architectures and alleges that its patents cover and claim such an architecture. I disagree with Patent Owner’s characterization. As I mentioned above, the ’454 patent never even

mentions the word “thread,” “multi-threaded,” “thread context,” “program context,” or anything else that may even remotely relate to or indicate a multi-threaded architecture. A POSITA would not understand the ’454 patent to relate to a multi-threaded unified shader. Rather, a multi-threaded unified shader would be an undisclosed improvement over what is disclosed in the ’454 patent. An example of this improvement would be the Lindholm Patents.

164. Many processors at the time of filing in 2003 employed a pipelined multi-threaded architecture. These architectures have been in use since the 1970’s, but the most well-known example processor is the Intel i860. This is the same processor used in Amanatides’ unified shader graphics architecture. This architecture renders the ’454 patent claims invalid.

1. Combining Amanatides and Kohn

165. A POSITA would be motivated to combine Amanatides with Kohn because both references use the Intel i860 processor and describe processing graphics with that same processor. The Kohn reference provides details regarding the same Intel i860 processor used in Amanatides. One of ordinary skill in the art would have been specifically motivated to combine the two references because they are directed to the same processor.

166. Moreover, Kohn states that the designers of the i860 processor “recognized a synergistic opportunity to incorporate a 3D graphics unit that

supports interactive visualization of results.” Ex. 1008 at 15. The i860 processor also contains “hardware necessary to implement transparent caching for multiprocessor systems moved costs beyond the silicon budget of this implementation.” *Id.* at 25. Like Amanatides, a POSITA would recognize that the i860 processor is ideal for implementing the flexible multiprocessor graphics architecture described in Amanatides. “After exploring several DSP microprocessors, we decided on the Intel i860. It is a powerful general-purpose microprocessor optimized for graphics.” Ex. 1007 at 4.

2. Independent Claim 2

a. Claim 2 [preamble] – Unified Shader

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

168. Amanatides discloses a unified shader graphics processing system that can load balance among different types of graphics data. In its unified shader design, Amanatides can employ the same processing unit for vertex and pixel processing while adjusting the number of processing resources for each type of graphics data.

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

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

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

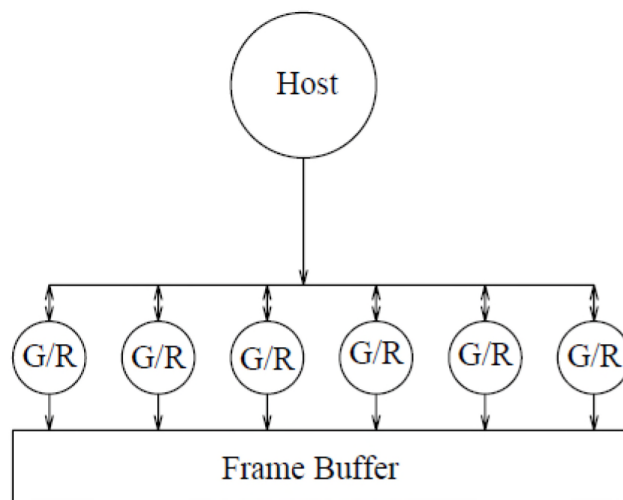


Figure 3

b. Claim 2[a] – General Purpose Register Block

170. Amanatides discloses a DRAM that stores both vertex and pixel data and is operatively coupled to the processor unit:

The new design would require more memory for each processor; a lot of memory is needed as textures and geometric models, if they are stored locally, are space intensive. As well, program space needs to be larger as each processor does more work. To keep costs down, we wanted to eliminate the need for SRAM, and use DRAM instead.

Ex. 1007 at 158. This is also depicted in Figure 4:

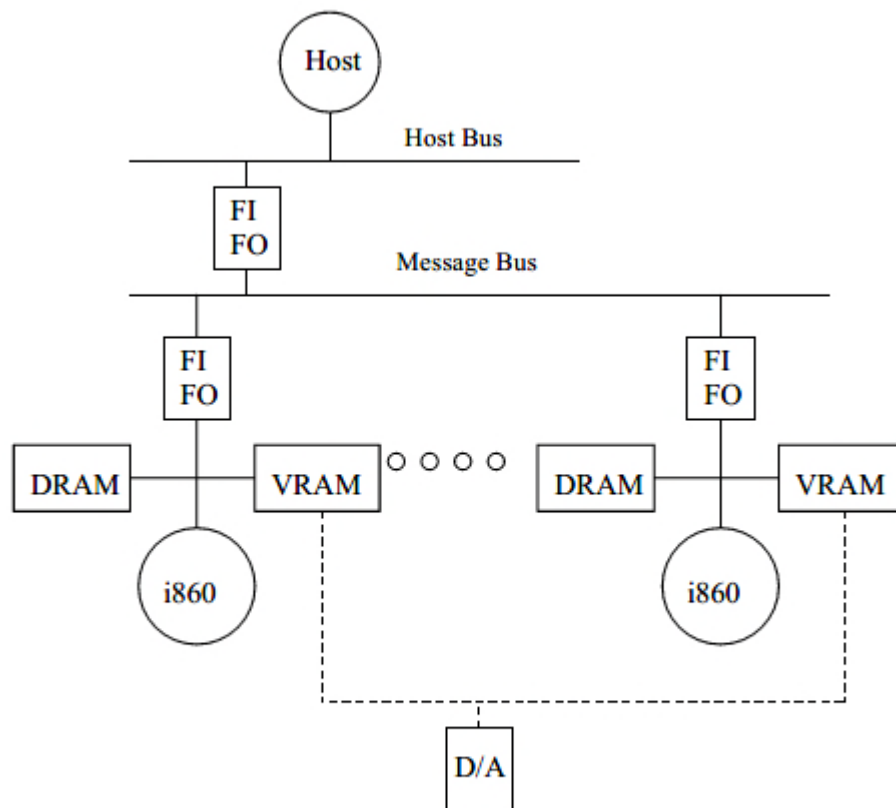


Figure 4

Ex. 1007 at 158. Amanatides also discloses 2 MB of VRAM associated with each i860 processor. Ex. 1007 at 157-58. Amanatides further discloses a bus and an input FIFO associated with each processor that store both vertex and pixel data:

There are two types of packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. As well, there is a fair scheduling policy in which a second packet from any FIFO cannot get onto the bus until all other FIFOs are first given a chance to send theirs.

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

Ex. 1007 at 158-59.

171. Both vertex data and pixel data are stored in an on-chip cache inside the i860 processor. *Id.* See also Ex. 1008 at 15 (“we divided the chip area evenly between blocks for integer operations, floating point operations, and instruction and data cache memories”), Fig. 2 (showing the “data cache”). As a result, each of these components comprises registers that can store both vertex and pixel data, and individually or jointly represent “a general purpose register block” as required by claims 1 and 2 and the data “store” of claim 5.

c. Claim 2[b] and [d] – Processor Unit

172. Claim elements 2[b] and [d] are disclosed by each of Amanatides and Kohn and the combination. The claimed “processor unit” can be interpreted as each G/R illustrated in Amanatides (the i860 processor). Ex. 1007. Alternatively, the “processor unit” can be construed as the functional units within the i860 processor (specifically, the floating point control unit, adder unit, and multiplier unit, along with the 3D graphics integer unit, either collectively or independently,

each capable of performing both vertex and pixel operations as elaborated herein).

Ex. 1008 at 15, 20, 24.

173. Amanatides discloses that in its G/R processors, the “front end” of graphics processing (vertex or “geometric” operations) and the “back end” (pixel or “rendering” operations) occur simultaneously in the same G/R processor:

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

Id. at 156-57.

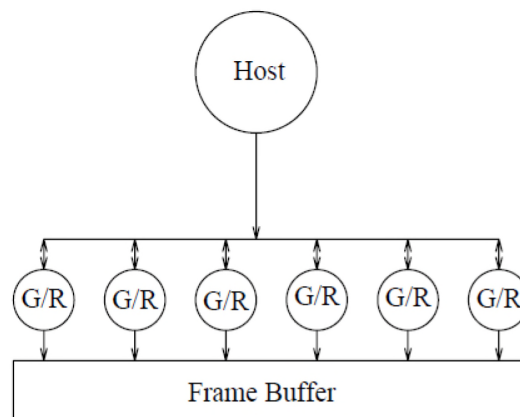


Figure 3

Ex. 1007 at Fig. 3.

174. The ‘454 patent describes a pipelined processor design with logical partitions to process both vertex and pixel operations. In essence, the ‘454 patent can handle multiple data types simultaneously due to its pipelined architecture. *Id.*

This fact is corroborated by the specification's description of processor 96 within the unified shader:

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

Ex. 1001 at 4:37-48. While a multi-threaded architecture emerged as an alternative means to achieve simultaneous processing of multiple data types (refer, e.g., to the Lindholm Patents), the '454 patent depended on partitioned segments or phases of a pipelined processor for the simultaneous processing of vertex and pixel data.

175. But this concept was not novel at the time of the invention. A decade prior, Amanatides, when read with the context of Kohn and the details of the i860 processor, had accomplished simultaneous processing of vertex and pixel data. As explained in Kohn, the i860 processor, employed in Amanatides as a unified shader, could carry out parallel execution of vertex and pixel data types. Ex. 1008

at 16-17. For example, Kohn describes that the floating-point instruction units of the i860 processor are capable of “efficiently perform[ing] the transformation calculations and advanced lighting calculations required for 3D graphics,” which are vertex operations. Ex. 1008 at 23-24. However, the floating-point units also support scalar operations, which as disclosed in the ‘454 patent, are characteristic of pixel operations. Ex. 1001 at 4:29-51; Ex. 1008 at 16, 22-23. Meanwhile, the “graphics unit” in the i860 could more quickly perform “back-end rendering operations” such as “shading,” which are pixel operations. Ex. 1008 at 24. But the “graphics unit” could also perform calculations on the vertices during transformation, *i.e.*, vertex operations. Ex. 1008 at 24. In this way, each i860 processor in the Amanatides architecture could “auto load balancing” between vertex and pixel work (G/R). Ex. 1008 at 157-58. In Amanatides' load-balancing architecture, the i860 processor evolved into a unified shader that could alternate between vertex and pixel processing depending on the data type present in the registers.

176. Additionally, both Amanatides and Kohn reveal (and a POSITA would have recognized) that the execution of instructions for geometry (vertex) operations culminates in the generation of position and appearance data, while the execution of instructions for rendering (pixel) operations determines the color of a pixel. *See* Ex. 1007 at 157-59; Ex. 1008 at 24.

177. Amanatides also discloses that its processor units perform geometry/vertex operations *in response to* receiving “geometry messages” with vertex data (GMs) and that rendering/pixel operations are performed *in response to* receiving “rendering messages” with pixel data (RMs). Ex. 1007 at 159-160.

d. Claim 2[c] - Sequencer

178. Both Amanatides and Kohn disclose the claimed “sequencer.” Amanatides features at least an on-chip cache for storing program instructions for vertex and pixel calculation operations. Ex. 1007 at 157-58 (“As well, program space needs to be larger as each processor does more work. To keep costs down, we wanted to eliminate the need for SRAM, and use DRAM instead. DRAM also has the advantage that a lot more memory can be placed per unit area. Unfortunately, DRAM is slower. To get back the speed, we wanted the processor to have an on-chip cache.”). The on-chip cache, together with the Memory Management Unit (Ex. 1008 at 15, 17, 20), comprise the claimed “sequencer” and sequencer “circuitry” of the ’454 patent.

179. Kohn offers additional insights regarding the on-chip instruction cache, as depicted in Fig. 2. Ex. 1008. Kohn provides detailed descriptions of the “instruction cache,” including its dual instruction mode. *Id.* at 16, 24-25. Moreover, Kohn discloses the capability to issue up to three operations per clock cycle, facilitating parallel execution by the various functional units within the

processor. *Id.* The instructions stored within the sequencer would encompass the necessary instructions for executing both vertex and pixel operations. The unified shader architecture described in the i860 processor (G/R processor) entails the simultaneous execution of front-end (vertex) operations and back-end (pixel) operations. These operations are carried out by executing instructions on pixel and vertex data. *See supra* at claim 2[b] and [d].

180. Finally, the “sequencer” is “coupled” to the various memories that comprise the “general purpose register block.” *See supra* at claim 2[a]. For instance, the instruction cache is “coupled to” not only the data cache within the i860 processor, but also the DRAM, VRAM, FIFO, and bus associated with each i860 processor. Ex. 1007 at Fig. 4; Ex. 1008 at Fig. 2.

3. Independent Claim 3

a. Claim 3 [preamble] – Unified Shader

181. *See supra* at §2[preamble].

b. Claim 3[a] and [c] – Processor Unit

182. Amanatides discloses a multiprocessor system in which each i860 processor can be regarded as the “processor unit.” Ex. 1007. Alternatively, the functional units within the i860 processor, namely the floating point control unit, RISC integer unit, adder unit, and multiplier unit, along with the 3D graphics unit within the i860 processor, individually or collectively, can also be considered as

the “processor unit.” Ex. 1008 at 15, 20, 24. Each of these processor units carries out both vertex and pixel calculation operations. *See supra* at §2[b] and [d].

183. Amanatides discloses that its i860 processors and the processor units within those processors can perform either vertex or pixel calculation operations based on a priority, employing automated load balancing. Amanatides discloses how its innovative “minimalist” unified shader architecture seamlessly integrates the vertex processing, which constitutes the “front end” of the graphics pipeline, with the pixel processing of the “back end”:

Here, the functions of the front and back ends are collapsed onto the same processors. In this case, the host distributes primitives in a round-robin manner to each of the processors. Each processor performs the front-end [vertex] 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. Because the primitives are distributed in this manner, each processor does the same back-end operations as in previous designs but only 1/*n*th of the front-end operations (assuming there are *n* processors).

Ex. 1007 at 157. Amanatides proceeds to outline several advantages of the system, one of which is that the processors can allocate a greater portion of their time to vertex (front end) processing or pixel (back end) processing, depending on

the amount of vertex work or pixel work that needs to be executed, *i.e.*, the data stored in the FIFO, DRAM, VRAM, and on-chip data cache of each processor:

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

Id. This 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 (*i.e.*, the quantity of vertex data or pixel data awaiting processing) or the available memory for each type of task. Similarly, Kohn observes that the floating-point and integer processor units in the i860 processor consisted of multiple stages, implying that different operations selected for the pipeline may be at different stages of execution. Ex. 1008 at 22-23. For instance, Kohn describes the parallel execution of operations in a “dual-operation mode” within the “pipelined mode,” where both floating-point operations and scalar operations could “take advantage of the pipelining capability.” *Id.* A POSITA would comprehend, based on the combination of Amanatides and Kohn, that the functional units within each i860 processor can concurrently perform both vertex operations and pixel operations, potentially at

different stages of completion, by switching between instructions stored in the instruction cache.

184. Amanatides also utilizes a priority allocation system, as described in claims 3 and 4. This system involves assigning a priority to either pixel or vertex data. It ensures that all pixels are processed until none remain or until the processor reaches its capacity for storing additional pixels. Similarly, vertices are processed until none remain or until the processor cannot store more vertices. Amanatides accomplished this by controlling the submission of vertex processing tasks (GMs) and pixel processing tasks (RMs) to the unified processor units via the bus.

There are two types of packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. As well, there is a fair scheduling policy in which a second packet from any FIFO cannot get onto the bus until all other FIFOs are first given a chance to send theirs. ...

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

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

Ex. 1007 at 158-59. Either GMs or RMs could be assigned “high priority.”

Additionally, the allocation of storage for vertex and pixel data is determined by the assigned priority for each data type.

185. In addition, Amanatides discloses that the FIFOs can be implemented in bidirectional FIFO chips, which can be programmed to “raise signals at various levels of filling.” Ex. 1007 at 158. Based on the disclosure, Amanatides describes “processor units” that prioritize or load balance either vertex operations or pixel operations at varying stages of completion, taking into account factors such as the workload requirements and available memory or shared resources for each type of processing. Consequently, the combination of Amanatides and Kohn, either individually or collectively, satisfies or renders obvious the limitations of the processor unit in claim 3, as well as the workload-related limitations in claims 1, 4, 5, and 11, which pertain to the processor unit and sequencer/instruction store.

c. Claim 3[b] – Shared Resources

186. I incorporate by reference the prior discussion at §2[a]. The “shared resources” mentioned include the “general purpose register block” (‘454 patent at 5:32-52). In Amanatides, this can encompass components such as the bus, FIFO, DRAM, and VRAM for each i860 processor, as discussed earlier. Additionally, in Kohn, various resources such as the on-chip instruction and data cache, memory management unit, bus control unit, core registers, floating-point registers, buses,

and other circuitry within each processor can be included. All these resources are “shared” for both vertex and pixel processing. The bus, FIFO, DRAM, and VRAM are operatively coupled to the i860 processor, while those components, along with the on-chip instruction and data cache, memory management unit, bus control unit, core registers, floating-point registers, buses, and other circuitry, are operatively coupled to the functional units within the processor (which can be individually or collectively considered the "processor unit" of the claim). *See supra* at §§2[a], [b] and [c].

4. Independent Claim 4

a. Claim 4 [preamble] – Unified Shader

187. *See supra* at §2[preamble].

b. Claim 4[a] and [c] – Processor Unit

188. *See supra* at §3[a] and [c].

c. Claim 4[b] – Shared Resources

189. *See supra* at §3[b].

5. Independent Claim 5

a. Claim 5 [preamble] – Unified Shader

190. *See supra* at §2[preamble].

b. Claim 5[a] – Processor Unit

191. *See supra* at §2[b].

c. Claim 5[b] – Sequencer

192. *See supra* at §§2[c], 3[a] and [c].

6. Dependent Claim 6

193. Dependent claim 6 recites, “The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memory.” As explained *supra* at §2[c], the “sequencer” in Amanatides includes the on-chip instruction cache. As described by Kohn, the instruction cache in the i860 processor is connected to a Memory Management Unit (MMU). Ex. 1008 at Figure 2, p. 17. The MMU translates virtual addresses (used by the software) into physical addresses (used by the hardware). “The address translation process maps virtual address space onto actual address space...” *Id.* at p. 20. The physical addresses are issued to the external memory via a 32-bit External Address bus. The data from the external memory is received via a 64-bit External Data bus via the Bus Control Unit and stored in the Data Cache for processing. *Id.* at Figure 2, p. 17. Accordingly, the circuitry of the i860 processor satisfies the claim limitations of “sequencer” and “further ... circuitry operative to fetch data from a memory.”

7. Dependent Claims 7 and 10

194. Dependent claim 7 recites, “The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a

control signal.” Dependent claim 10 recites, “The shader of claim 7, wherein the control signal is provided by an arbiter.”

195. As discussed earlier in claim 3[a] and [c], the combination of Amanatides and Kohn encompasses an arbiter responsible for determining the input (*e.g.*, vertex or pixel) to be provided to the processor units. Amanatides incorporates a priority allocation system in which one type of graphics data, such as pixels, is assigned priority. This ensures that pixel samples are processed until none remain or until the processor exhausts its storage capacity for additional pixel samples. Moreover, specific packets, whether vertex or pixel, can be designated with high or low priority. According to Amanatides, “low priority packets cannot get onto the bus if high priority packets are waiting.” Ex. 1007 at 158-59.

Therefore, 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, such as the designation of high or low priority for the packets, as further elaborated below.

196. Amanatides refers to assigning low-priority and high-priority packets to the time slots of the bus as “arbitration.” Ex. 1007 at 159. Additionally, the FIFOs can be programmed to “raise signals at various levels of filling.” *Id.* Amanatides describes the packets transferred between processors on the message-passing bus (“THE BUS”). *Id.* According to Amanatides, “There are two types of

packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. As well, there is a fair scheduling policy in which a second packet from any FIFO cannot get onto the bus until all other FIFOs are first given a chance to send theirs.” *Id.* “Running at 10MHz, 2 time slots are required for arbitration.” *Id.* In this case, the “arbitration” occurs on the message passing bus, which would be performed by logic circuits that are not explicitly described in Amanatides but would be recognized by a POSITA as being necessarily present. The “control signal” is part of the packet itself, such as indicating whether it is of “low” or “high” priority.

8. Dependent Claim 8

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

9. Dependent Claim 9

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

10. Independent Claim 11

a. Claim 11 [preamble] – Unified Shader

199. *See supra* at §2[preamble].

b. Claim 11[a] and [c] – Processor Unit

200. *See supra* at §§2[c], [d], 3[a], [c].

c. Claim 11[b] – Instruction Store

201. The “instruction store” is met by the same components that satisfy the “sequencer” limitation of claim 2. *See supra* at §§2[c], [d], 3[a], [c]. 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 various stages of execution. Ex. 1008 at 22-23. For instance, Kohn describes the execution of operations in parallel, specifically in a “dual-operation mode” within the “pipelined mode,” whereby both floating-point operations and scalar operations could “take advantage of the pipelining capability.” *Id.* A POSITA would, upon considering the combination of Amanatides and Kohn, comprehend 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 the switching between instructions stored in the instruction cache.

11. Independent Claim 1

202. Claim 1 is a method claim that merely puts the limitations of apparatus claim 3 into method claim format. For example, the steps to be

performed in claim 1 align with the functional elements of the processor unit, and the availability of space in the general-purpose register block in claim 1 (which stores both vertex and pixel data) aligns with the availability of shared resources in apparatus claim 3. As discussed in claims 2[b] and [d], in Amanatides' load-balancing architecture, the i860 processor evolved into a unified shader that could alternate between vertex and pixel processing depending on the data type present in the registers. Ex. 1008 at 157-58. This implies that the processor would not process incoming vertex or pixel data unless there was enough space in the registers to store this data. A POSITA would understand that load-balancing does not only relate to the processing but also to the storing of the associated vertex or pixel data, *i.e.*, processing a certain data type (vertex or pixel) cannot proceed unless that data type can be stored in the registers. As discussed in connection with claim limitations 3[a] and [c], priority can be given to pixel operations or vertex operations in the Amanatides system, such that vertex operations could be given priority as long as there are sufficient shared resources (including space in the general-purpose register block) for vertex processing, and when there is not, prioritizing pixel operations. Accordingly, the limitations of claim 1 are at least obvious based on the combination of Amanatides and Kohn and are met for the same reasons as claims 2 and 3.

XI. GROUND #3: CLAIMS 1-11 BASED ON THE COMBINATION OF SELZER AND FISKE

203. I have reviewed and am familiar with the following prior art references:

- a. Harald Selzer, Dynamic Load Balancing within a High Performance Graphics System, In Proceedings of Rendering, Visualization and Rasterization Hardware (Eurographics' 91 Workshop) at 37-53 (Springer-Verlag 1993) (“Selzer”) was published in 1993. Ex. 1009.
- b. Stuart Fiske and William J. Dally, Thread prioritization: A thread scheduling mechanism for multiple-context parallel processors, In Proceedings of First Symposium on High-Performance Computer Architecture, 1995 at 210-221 (IEEE 1995) (“Fiske”) was published in 1995. Ex. 1010.

204. It is my opinion that the combination of Selzer and Fiske renders claims 1-11 of the '454 Patent obvious.

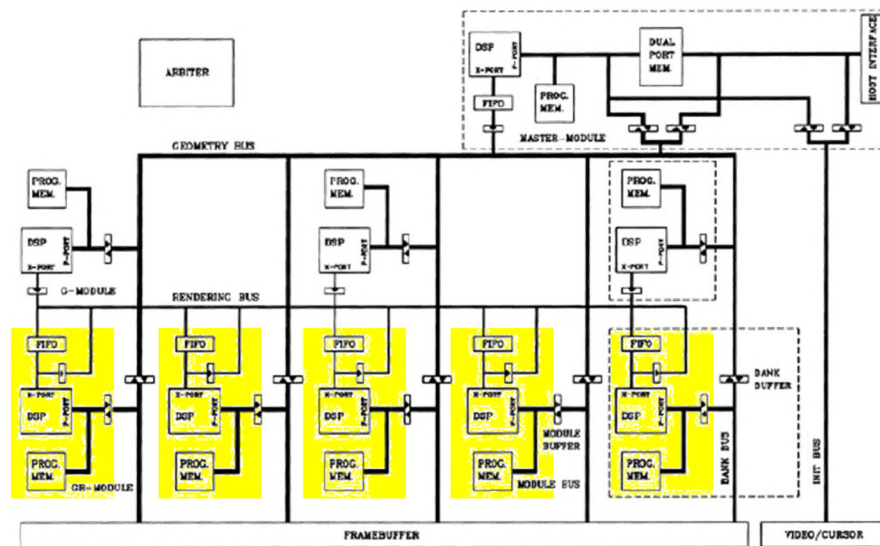
A. Selzer

205. The Selzer paper, titled “Dynamic Load Balancing within a High Performance Graphics System,” discloses a unified shader graphics architecture capable of dynamic load balancing between vertex and pixel operations. Such architecture is well-suited for interactive 3D graphics applications, which require significant levels of arithmetic processing to meet demands for high image complexity and resolution.

206. Selzer's processor architecture is connected to a distributed frame buffer and incorporates a significant amount of parallelism; that is, it "can be configured with an arbitrary number of identical, high level programmable processors operating in parallel." The architecture's load balancing mechanism automatically distributes the processing load between vertex operations and pixel operations, depending on the size and position of the geometrical objects being processed.

207. Each Rendering Module in Selzer's architecture can perform vertex operations (geometry) and pixel operations (rendering). As Selzer describes, "[t]he geometry module transforms and clips the graphic primitives, subdivides biparametric patches and the lighting calculations that are necessary and tasks like this. The rendering module performs the shading algorithms and transfers pixel data to the frame buffer. The rendering module also supports too all functions of the geometry module (Figure 4.1)." Ex. 1009 at 4.

208. Graphics data fed through Selzer's system is processed simultaneously and is then transferred to subsequent modules in parallel. Selzer's system employs a significant degree of parallelism to make this process run smoothly. One of Selzer's diagrams depicts a processor with five rendering modules and three geometry modules:



Ex. 1009 at 8. Indeed, this system is highly flexible, and Selzer explicitly states that “[p]arallelism should be implemented wherever possible.” This is like the claims in the ‘454 patent, which repeatedly discuss operative coupling, simultaneous activity, and other features indicative of parallel structure.

209. Selzer's parallelism requires the system to focus on load balancing between vertex and pixel operations to maintain peak efficiency throughout processing. Selzer notes that:

“[t]he effort of computation in the geometry and rendering section depends on size and position of the geometrical objects. ... With the intention of exploiting all the distributed computational power of the system, the processing units have been able to adapt their activities to the actual processing requirements of the scene. To enable such a dynamic load balancing and to speed up geometry calculation dynamically if required, the rendering modules are capable of performing all

the geometry calculations too. After rendering an object any processor may run into an idle state if there is no rendering data in his input buffer. Performing a task switch it will request new unprocessed geometry objects and continue geometry calculations. In this way an automatic load balancing is achieved across all the processors. When several rendering modules are doing geometrical calculation the overall rendering performance is reduced in favour of geometry processing power. Doing so the exploitation of the processing power incorporated encounters more than 95% and no computational power is going to be wasted by a rendering module starting out to run an idle state.”

Ex. 1009 at 10-11.

210. With respect to task switching, Selzer provides that “[t]he capability of automatically distributing the workloads between geometry and rendering modules means inherently task switching between two jobs within the same application. Supported by the large local memory (up to 256k x 32bit) the switching is reduced to saving and restoring of all processor registers, processing interrupt control and inspecting the rendering FIFO status. This is performed in less than 5 us. Since this time is very small with respect to the time consumed for geometry processing the self balancing is even useful for scenes comprising only very small triangles.” Ex. 1009 at 11-12.

211. Peak performance of Selzer's architecture occurs "when all rendering processor power is exploited for rendering calculations and the rendering are working continuously. If the balancing mechanism is activated the peak performance is not achieved, but the processing power of the rendering modules is used to speed up geometry calculations. This has two effects: - it prevents the rendering modules from running into an idle state and - speeds up image generation by supporting substantially geometry processing." Ex. 1009 at 12-13.

212. Selzer is therefore a unified shader graphics processor that uses parallel processing to enable simultaneous execution of multiple graphics data types. Selzer addresses the same problem as the '454 patent—preventing underutilization of resources when processing vertex and pixel graphics data. In Selzer's system, "the processing units have to be able to adapt their activities to the actual processing requirement of the scene." Ex. 1009 at 46. Thus, Selzer solves this problem with processing units that can perform processing on different types of graphics data. The Selzer system includes "geometry modules," each of which "transforms and clips the graphic primitives, subdivides biparametric patches and the lighting calculations that are necessary," and "rendering modules," each of which "performs the shading algorithms and transfers pixel data to the frame buffer." *Id.* at 40. Importantly, "if required, the rendering modules are capable of performing all the geometry calculations too." *Id.* at 46-47. Selzer achieves

dynamic load balancing among different graphics tasks through its flexible processing system, namely vertex and pixel processing.

B. Fiske

213. The Fiske paper, titled “Thread Prioritization: A Thread Scheduling Mechanism for Multiple-Context Parallel Processors,” discloses a multiple-context processor that dynamically changes the priority of each thread to balance the workload for various tasks and datasets. Fiske’s processor “provide[s] register resources that allow rapid context switching between several threads as a means of tolerating long communication and synchronization latencies.” Ex. 1010 at Abstract. Fiske’s thread prioritization would improve cache performance and would be particularly useful “in making load balancing decisions, and operating system scheduling decisions.” Ex. 1010 at 220.

214. In circumstances where the system is processing diverse data types with varying priorities, Fiske teaches two levels of prioritization: the first, when vertex or pixel data is assigned to a thread context (also known as thread allocation), and the second, when instructions for threads are scheduled for execution (also known as thread execution). As Fiske explains:

When scheduling threads on such a processor, we must first decide which threads should have their state loaded into the multiple contexts, and second, which loaded thread is to execute instructions at any given time. In this paper we show that both decisions are

important, and that incorrect choices can lead to serious performance degradation. We propose thread prioritization as a means of guiding both levels of scheduling. Each thread has a priority that can change dynamically, and that the scheduler uses to allocate as many computation resources as possible to critical threads.

Ex. 1010 at Abstract.

215. Fiske observed that this prioritization process improved barrier benchmark performance by up to a factor of two, and improved queue lock benchmark performance by up to a factor of 7. Ex. 1010 at 211. He also noted that his system improved performance by up to 26% for the fine-grain synchronization benchmark. *Id.*

216. Fiske describes his automatic load balancing scheme as a general processor, but the concepts he discloses would also work well in a graphics processing system. In a system with only one data type to process, priority and load balancing would not be necessary. As Fiske explains:

Thread prioritization involves assigning a priority to all the different threads in an application, and then using this priority to make thread scheduling decisions. The priority reflects the importance of a single thread to the completion of a single application. The thread scheduler uses the thread priority in a very different way than process scheduling in UNIX for instance, where the goal is to achieve good interactive performance and time sharing between competing

processes [16]. In our case, 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. Also, the granularity of scheduling is much different: in our case the priority is used to make scheduling decisions on every hardware context switch in a multiple-context processor.

Ex. 1010 at 212.

217. The ‘454 patent employs similar prioritization. Claim 1 of the ‘454 patent provides that the processor operations are performed “based on the provided inputs to the unified shader,” and claim 8 includes an arbitration scheme “configured to generate a control signal for determining which one of the vertex data input and the pixel data input is to be transmitted to the unified shader.” Indeed, by relying on the control signal to identify which input to transmit to the shader, the ‘454 patent prioritizes threads similarly to Fiske.

218. The unified shader in the ‘454 patent also “has the 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, very quickly.”

Ex. 1001 at 5:31–35. Similarly, Fiske’s thread prioritization system allows Fiske’s processor to schedule instructions of prioritized threads for execution, switching at will between scheduling and processing each data type:

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. The software scheduler on each node chooses the highest priority threads as the loaded threads. 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. The goal is to devote as many of the processor resources as possible to the tasks that are known to be critical to overall performance.

Ex. 1010 at 212.

[A]t each context switch, the hardware scheduler uses the priority to determine which loaded thread to execute... If a thread is waiting for a memory reference to be satisfied then it is *stalled* and is not considered for scheduling until the memory reference is satisfied. If several loaded threads have the same priority, then these threads are chosen in round-robin fashion. A context switch can occur on a cache miss, on a failed synchronization test, or on a change of priority on one of the threads in the processor.

Ex. 1010 at 211.

219. Consequently, Fiske's disclosure encompasses a multithreaded processor incorporating register resources to facilitate swift context switching between multiple threads. Fiske's system is specifically engineered to be dynamic and utilizes a thread prioritization scheme to facilitate a flexible approach to thread

scheduling. Ex. 1010 at 210-11. As Fiske offers multiple contexts, enabling the simultaneous loading of multiple threads, multiple threads are always present and ready to run at any given time. *Id.*

220. Fiske emphasizes that the scheduling of threads and the efficiency of a multithreaded processor involve a two-step consideration. First, Fiske's system determines which threads should be loaded into one of the available contexts. *Id.* at 210. Second, Fiske's system employs a thread prioritization scheme among the threads loaded into the context to determine which thread's instructions are executed. *Id.* at 211. Fiske acknowledges that the runtime performance would be negatively impacted if all threads within the system were treated equally, without any prioritization. Therefore, depending on the processor's workload, different threads can be selected at different times based on the employed priority scheme. *Id.* at 210-11.

C. Combining Selzer and Fiske

221. In my opinion, a POSITA would have found it very straightforward and obvious to combine the teachings of the unified shader of Selzer to use the multithreaded architecture of Fiske.

222. Both Selzer and Fiske are directed dynamic load balancing within a processing system. A POSITA would recognize this important similarity and readily understand that these two references are directed to solving the same

purported problem of inefficiency in a processing system. A POSITA would further understand that the only difference between the approaches of Selzer and Fiske is that Fiske incorporates multiple hardware thread contexts to allow for rapid switching among threads, whereas Selzer utilizes a scalable approach to parallelism by multiplying geometry and rendering modules.

223. Combining the architectures of Selzer and Fiske would have yielded predictable results. For example, the exemplary configuration of Selzer's architecture is three geometry and five rendering modules. *Id.* at 43. Selzer further notes that its system is highly flexible and the number of modules can be easily adjusted. *Id.* Each module is responsible for performing operations on a particular graphics thread (vertex or pixel), and the rendering modules can perform both vertex and pixel processing. This flexibility (and unified shader approach) allows for dynamic load balancing because the rendering modules can switch from pixel tasks to vertex tasks at any time and load balance the system. A POSITA would understand that Fiske discloses the same type of dynamic load balancing to keep its system operating efficiently, but Fiske condenses it into a single processing module that includes multiple thread contexts. A POSITA would recognize that this combination would perform the exact same operations as Selzer's numerous rendering modules but would allow for savings in chip space.

224. Once a POSITA combines the multiple thread contexts of Fiske into a rendering module of Selzer, the same POSITA would know that Fiske's scheduling approach could be utilized in the new combination. Utilizing Fiske's scheduling approach, the system could dynamically adjust to allow for additional register space for vertex or pixel threads depending on the workload. Then, based on the number of loaded pixel or vertex thread contexts, Fiske's approach would further dynamically schedule a thread for execution. This is very similar to the Selzer scheduling approach, where the rendering module would prioritize pixel processing ahead of vertex processing. In the combined system, the scheduling unit could designate additional thread contexts to pixel threads, and schedule them for processing.

225. A POSITA would possess the knowledge that any multithreaded architecture would include decision-making functionality for thread assignment/allocation and thread execution, as those decisions must be made in any system. This concept is also reflected in Fiske. *See* Ex. 1010 at 210-11. Accordingly, a POSITA taking advantage of the known benefits of the unified shader of Selzer with the additional benefits of a multithreaded architecture would necessarily include such decision-making functionality. *See id.*

226. And although Selzer and Fiske are clear that prioritization of operations (and threads) was important in the prior art, a POSITA reading Fiske,

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

1. Independent Claim 2

a. Claim 2 [preamble] – Unified Shader

227. I understand that the preamble may not be a limitation to claim 2. However, if it is, Selzer discloses a graphics processor with a unified shader either alone or in combination with Fiske. Selzer describes “a graphics processor architecture with a high degree of parallelism connected to a distributed frame buffer. ... Within the architecture an automatic load balancing mechanism is presented which distributes the processing load between geometry and rendering section.” Ex. 1009 at 37. Selzer’s architecture includes a “rendering module” that “performs the shading algorithms and transfers pixel data to the frame buffer. The rendering module also supports too all functions of the geometry module.” *Id.* at 40-41. Because the rendering module performs both vertex and pixel operations, it is a unified shader according to the ’454 patent.

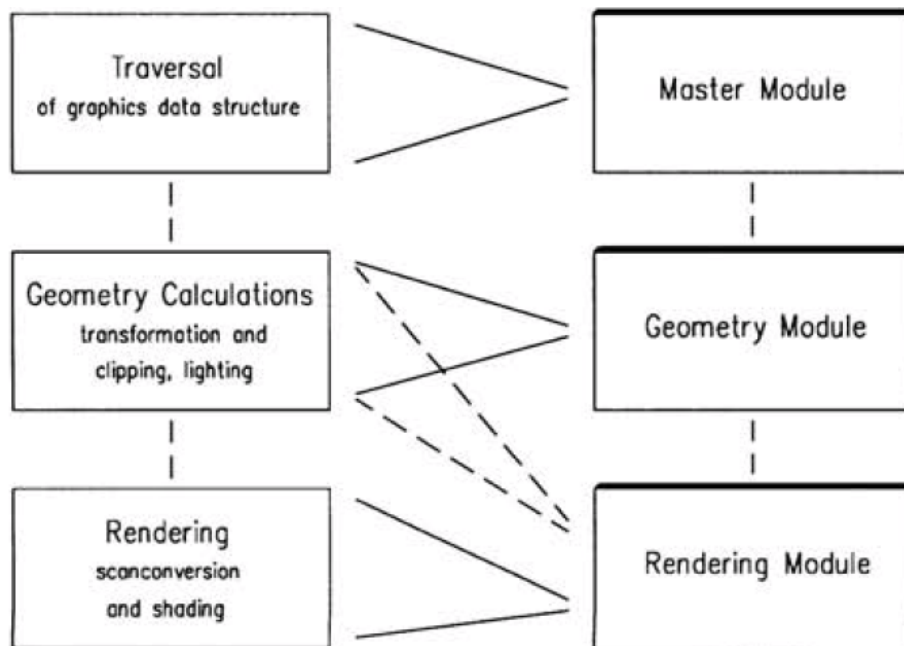


FIGURE 4.1. Mapping functional sections to hardware

228. It is my opinion that the claims of the '454 patent do not require a multithreaded processor, but even if that were found to be necessary, the combination of Selzer and Fiske teaches a multithreaded unified shader. As I noted above in paragraphs 221-226, a POSITA can easily implement Fiske's multithreaded processing architecture as a rendering module in Selzer. And moreover, a POSITA, as I noted above, would have been motivated to do so based on the teachings of Selzer and Fiske to achieve the architectural efficiencies of both systems. In other words, while both systems are directed to efficiently perform multiple operations, a POSITA would recognize that by combining the two architectures, those efficiencies noted by Fiske could be readily implemented

in the flexible graphics system of Selzer. Utilizing the multiple thread contexts of Fiske would allow for a single rendering module in Selzer. This combination rendering module with multiple thread contexts (i.e., a multi-threaded processor unit) would allow for the processing of both vertex and pixel operations and further allow for dynamic scheduling and load balancing of the threads. Thus, claim 2 is met by the combination of Selzer and Fiske.

b. Claim 2[a] – General Purpose Register Block

229. Selzer discloses that each of its rendering processors and the master module includes general register blocks in the form of programmable memories and FIFOs that receive and maintain all geometry (vertex) and rendering (pixel) data. Ex. 1009 at 44-45.

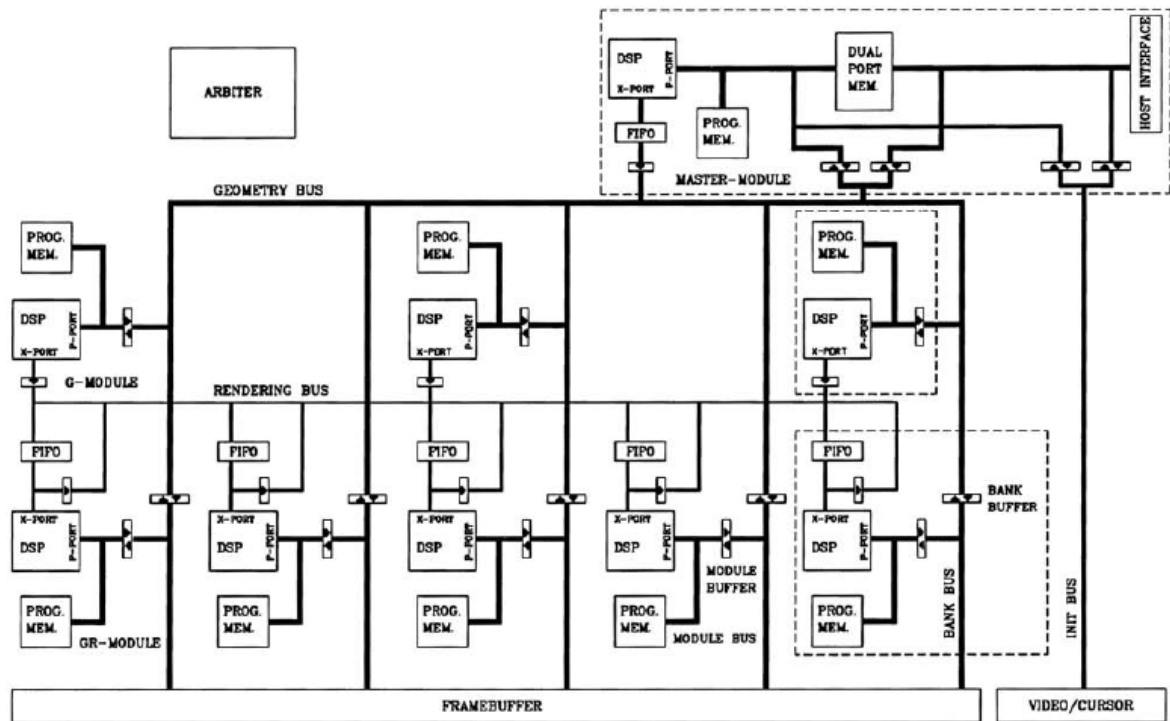


FIGURE 4.4. System architecture

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

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

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

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

c. Claim 2[b] and [d] – Processor Unit

232. In Selzer, a rendering module can process both vertex and pixel operations. Ex. 1009 at 41-43.

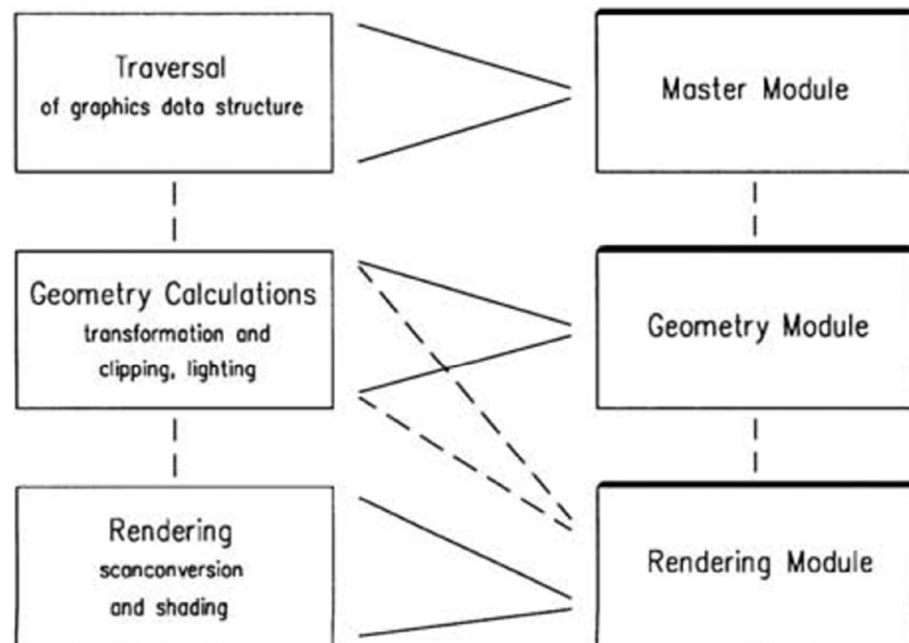


FIGURE 4.1. Mapping functional sections to hardware

Id. at 41. Selzer describes this process in detail, but as shown in the figure above, both geometry calculations (vertex operations) and rendering (pixel operations) may be directed to the Rendering Module. *Id.* Because of this flexibility in

Selzer's system, the Rendering Module is a unified shader processor unit that can perform either vertex or pixel operations at any time.

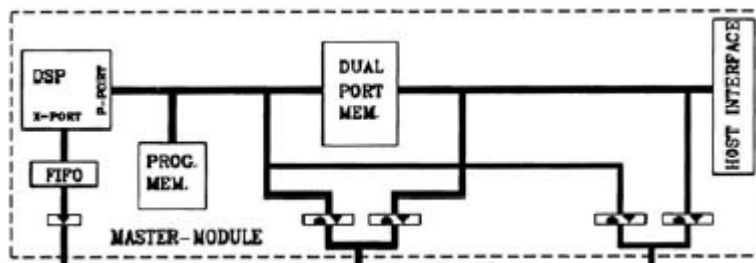
233. Claim 2 of the '454 patent recites a processor unit that executes instructions to generate a pixel color. The rendering module generates a pixel color and conveys the calculated pixel colors and values to the framebuffer. *Id.* at 42. The pixel data on which the rendering module performs the pixel operations is selected from the general-purpose register block (described above in connection with claim limitation 2[a]) under the coordination of the master module, and is generated in response to receiving the selected pixel data. *Id.*

234. The rendering module also performs what the '454 patent claim 2 refers to as generating vertex position and appearance data. That is, the operations the rendering module in Selzer may perform include transformation, clipping, polygon and patch subdivision, normal interpolation, and lighting operations. *Id.* These are all standard vertex operations that are involved in the generation of position and appearance. The vertex operations are performed based upon the vertex data and instructions that are selected from the general-purpose register block (described above in connection with claim limitation 2[a]) under the coordination of the master module, and is generated in response to receiving the selected vertex data. *Id.*

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

d. Claim 2[c] - Sequencer

236. Selzer discloses the claimed sequencer as recited in claim 2[c] of the '454 patent. The master module in Selzer accesses the graphics data in the system and “fetches the next primitive or task automatically.” Ex. 1009 at 43.



Id. at 44. The master module has access to various busses and the framebuffer and like the framebuffer maintains graphics data and its associated instructions to provide to the rendering module. *Id.* The master module coordinates the processing and load balancing in Selzer’s system, and Selzer describes the master module as the “systems supervisor” that “is responsible for start-up and synchronizing activities.” *Id.* at 40. The master module gathers the graphics data and tasks and supplies those tasks (data and instructions) to the appropriate processor unit. *Id.* at 41. As noted above, the master module can switch between sending vertex or pixel tasks to each rendering unit, allowing them to load balance

among graphics operations. Once calculated, the data is stored in the local memories on each unit and eventually sent to the framebuffer. *Id.*

237. To the extent a multithreaded processor unit is read into claim 2, Selzer combined with Fiske would teach a sequencer for a multithreaded processor unit. When Selzer's rendering module is combined with Fiske's scheduling module and multiple thread contexts, it teaches claim 2[c] in the context of a multithreaded processing unit.

238. As I described previously, the combination of Selzer and Fiske would place the multiple thread contexts of Fiske into a rendering module of Selzer. Given this combination, a POSITA would understand that the Fiske scheduling algorithm would choose the active thread for which the rendering module would execute the next instruction. Although Fiske describes this in detail, a POSITA would possess this fundamental knowledge of scheduling instructions in a processor that is inherent to any processor technology. In the Selzer/Fiske combination, the selected instruction for the thread would either be for a vertex calculation or a pixel calculation. 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.

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

2. Independent Claim 3

a. Claim 3 [preamble] – Unified Shader

240. *See supra* at §2[preamble].

b. Claim 3[a] and [c] – Processor Unit

241. Selzer's system includes a rendering module that prioritizes pixel calculations, which I discussed at length for claim 2. Ex.1009 at 46-47. Selzer's system is dynamic and can adjust to process either vertex or pixel operations in the rendering module, but in one example, Selzer describes how the rendering module prioritizes pixel operations, but if it reaches an idle state (i.e., no pixel resources are available), then it switches to perform vertex operations. *Id.* at 47. Selzer notes that while pixel processing performance is reduced in this situation, the gains in switching to perform vertex operations outweigh that loss and provide for a more efficient processing unit. *Id.*

242. Even though Selzer fully discloses all elements of claim 3[a] and 3[c], when combined with Fiske, vertex or pixel operations scheduling becomes even more fine-grained and dynamic. As explained above for claim 2, the scheduling algorithm in Fiske, with its multiple thread contexts, can load balance in two ways. First, if enough shared resources are unavailable for pixel processing, then the

Selzer/Fiske combination would allocate its thread contexts for vertex processing. Ex. 1010 at 210-211. Second, once those thread contexts are allocated, Fiske performs another selection to determine which resources should be used for vertex or pixel operations. *Id.* Fiske describes this process as being done in a fine-granted manner that can switch among different instructions of a thread and at a time when the entire thread is not completed. *Id.* In this manner, the combination of Selzer and Fiske would work in a multithreaded environment and meet claim 3.

c. Claim 3[b] – Shared Resources

243. I incorporate by reference the discussion *supra* at §2[a].

3. Independent Claim 4

a. Claim 4 [preamble] – Unified Shader

See supra at §2[preamble].

b. Claim 4[a] and [c] – Processor Unit

244. *See supra* at §3[a] and [c].

c. Claim 4[b] – Shared Resources

245. *See supra* at §3[b].

4. Independent Claim 5

a. Claim 5 [preamble] – Unified Shader

246. *See supra* at §2[preamble].

b. Claim 5[a] – Processor Unit

247. *See supra* at §2[b].

c. Claim 5[b] – Sequencer

248. *See supra* at §§2[c], 3[a] and [c]. In addition, as I described above for claim 3, the scheduling architecture in Fiske allows for dynamic allocation of processing resources to whatever data is present and ready for execution.

Accordingly, a POSITA would understand that the system in Fiske could readily prioritize vertex processing or pixel processing depending on the amount of memory space available for either operation and data, or depending on the processor resources available for each thread.

5. Dependent Claim 6

249. Dependent claim 6 recites, “The shader of claim 5, wherein the sequencer further includes circuitry operative to fetch data from a memorySelzer’s master module has direct access to the busses and fetches the appropriate data to supply to each rendering module. This meets claim 6.

6. Dependent Claims 7 and 10

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

251. Selzer discloses that the master module includes a DSP circuit, which decides (makes a choice among several option) of whether a rendering module

should receive vertex data for vertex operations or pixel data for pixel operations. Ex. 1009 at 43-45. The master module circuit then sends the appropriate data to the rendering module for it to perform either vertex or pixel operations. Thus, the DSP in the master module provides a control signal that the master module circuit then uses to tell the rendering module which types of operations it will perform. This meets claims 7 and 10.

252. In addition, the combination of Selzer and Fiske teaches that a circuit (*e.g.*, master module in Selzer, scheduling arbiter in Fiske) provides information (*i.e.*, data) to the store (*i.e.*, memories and FIFOs) in response to a signal dictating whether to process vertex or pixel data. Because the modules in Selzer and Fiske are deciding what data to select based on the variable status of the system (*e.g.*, the switching between vertex and pixel processing in the rendering modules in Selzer, or the context switching disclosed in Fiske), this process is performed by an arbiter. For example, the hardware scheduler in Fiske “chooses the loaded thread with the highest priority as the next thread using simple hardware, in order to minimize context switch overhead.” Ex. 1010 at 211. In this case, the “hardware scheduler” is the claimed arbiter, and the “simple hardware” within the arbiter provides the claimed control signal. Alternatively, Fiske discloses that the “software scheduler on each node chooses the highest priority threads as the loaded threads.” Ex. 1010 at 211. In this case, the “software scheduler” is the claimed

arbiter, and instructions inside the software scheduler provide the claimed control signal. The threads in Fiske are vertex or pixel threads that fetch data from a cache, the claimed “store.” “If a thread is waiting for a memory reference to be satisfied then it is stalled and is not considered for scheduling until the memory reference is satisfied.” Ex. 1010 at 211. “[E]very single memory reference is simulated through the cache [5, 9] since the hit rates for instructions and 'local data are typically very high.” *Id.* at 212. Accordingly, Fiske, in combination with Selzer, teaches dependent claim elements 7 & 10.

7. Dependent Claim 8

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

8. Dependent Claim 9

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

9. Independent Claim 11

a. Claim 11 [preamble] – Unified Shader

255. *See supra* at §2[preamble].

b. Claim 11[a] and [c] – Processor Unit

256. *See supra* at §§2[c], [d], 3[a], [c]. As I described above for claims 2, 3, 4, and 5, Fiske performs a selection to determine which resources should be used for vertex or pixel operations. *Id.* Fiske describes this process being done in a fine-granted manner that can switch among different instructions of a thread, and at a time when the entire thread is not completed. *Id.* This further meets the limitations of claim 11, because the multithreaded processor would perform vertex or pixel operations based on vertex or pixel workload, and can switch between instructions in the instruction store and perform vertex and pixel operations at various degrees of completion (*e.g.*, switch between unfinished vertex threads and unfinished pixel threads).

c. Claim 11[b] – Instruction Store

257. The “instruction store” is met by the same components that satisfy the “sequencer” limitation of claim 2. *See supra* at §§2[c], [d], 3[a], [c]. Selzer specifically refers to instruction storage and notes that the processors each include up to 256k*32 bit wide “fast static memory for instruction and data storage.” Ex. 1009 at 40.

10. Independent Claim 1

258. Claim 1 is a method claim that merely puts the limitations of apparatus claim 3 into method claim format. For example, the steps to be performed in claim 1 align with the functional elements of the processor unit, and

the availability of space in the general-purpose register block in claim 1 (which stores both vertex and pixel data) aligns with the availability of shared resources in apparatus claim 3. As discussed earlier in claims 2[b] and [d], in Selzer's architecture, the pixel or vertex data on which the rendering module performs the operations comes from the memory (general purpose register block) and/or the master module. Ex. 1009 at 40-44. The master module gathers the graphics data and tasks and supplies those tasks (data and instructions) to the appropriate processor unit. *Id.* at 41. As noted above, the master module can switch between sending vertex or pixel tasks to each rendering unit, allowing them to load balance among graphics operations. Once calculated, the data is stored in the local memories on each unit and eventually sent to the framebuffer. *Id.* A POSITA would understand that the master unit would be unable to send data to the processing unit if it could not be stored in its memory, local memories, or the framebuffer. Therefore, the processing of the data cannot proceed unless there is sufficient storage in the processor's general-purpose registers. As I discussed in connection with claim limitations 3[a] and [c], Selzer can dynamically prioritize vertex operations or pixel operations based on workload and the availability of resources, including space available in the general-purpose register block, such that vertex operations could be given priority as long as there are sufficient shared resources (including space in the general-purpose register block) for vertex

processing, and when there is not, prioritizing pixel operations. Accordingly, the limitations of claim 1 are at least rendered obvious by the combination of Selzer and Fiske and are met for the same reasons as claims 2 and 3.

XII. DECLARATION

259. In signing this declaration, I understand that this declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the USPTO.

260. I am aware that willfully false statements and the like made by me in connection with this declaration are punishable by fine or imprisonment, or both (18 U.S.C. § 1001) and may jeopardize the validity of the Petition. All statements herein made by me of my own knowledge are true and all statements made on information and belief are believed to be true. I declare under penalty of perjury that the foregoing is true and correct.

Executed this 19th day of May 2023, at 6 Pioneer Rd., Arlington, MA
02474, USA.

Respectfully submitted,

A handwritten signature in dark ink, appearing to read "Hanspeter Pfister". The signature is written in a cursive, flowing style.

Hanspeter Pfister, Ph.D.