

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. IPR2023-00564
U.S. Patent No. 7,742,053

**DECLARATION OF JOHN HART REGARDING
U.S. PATENT NO. 7,742,053**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. QUALIFICATIONS AND EXPERIENCE.....	2
III. MATERIALS CONSIDERED	4
IV. LEGAL STANDARDS	6
A. Claim Construction.....	7
B. Obviousness.....	7
V. RELEVANT TECHNICAL BACKGROUND AT THE TIME OF THE '053 PATENT	9
VI. U.S. PATENT NO. 7,742,053	11
VII. DESCRIPTION OF ASSERTED ART.....	13
A. Stuttard (PCT Pub. No. WO00/62182A2).....	13
B. Williams (PCT Pub. No. WO00/63770A1).....	17
C. Whittaker (U.S. Patent No. 5,968,167)	18
VIII. CLAIM CONSTRUCTION	18
A. Storage of “command threads”	21
B. “at least one memory device . . . operative to store a plurality of pixel command threads and . . . operative to store a plurality of vertex command threads”	23
C. “select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads”	24
D. “command processing engine”	25
IX. EVALUATION OF PETITIONER’S PROPOSED GROUNDS	27
A. Ground 1: Stuttard and the Knowledge of a POSITA Does Not Render Claims 1-9 of the '053 Patent Obvious	27
1. All Claims: Stuttard Does Not Disclose or Render Obvious the Claimed “at least one memory device”	28
2. All Claims: Stuttard Does Not Disclose or Render Obvious Storing Complete Command Threads	41

TABLE OF CONTENTS
(Continued)

	<u>Page</u>
3. All Claims: Stuttard Does Not Disclose Or Render Obvious An Arbiter That Selects A Command Thread From Either Of The Plurality Of Pixel Command Threads And The Plurality Of Vertex Command Threads	43
4. Claim 7: Stuttard Does Not Disclose Or Render Obvious A Texture Processing Engine.....	46
B. Ground 2: Stuttard, Williams, and the Knowledge of a POSITA Does Not Render Claims 1-9 of the '053 Patent Obvious.....	47
1. All Claims: Williams Does Not Disclose Or Render Obvious “Command Threads”	47
2. All Claims: Williams Does Not Disclose Or Render Obvious Storing Vertex and Pixel Command Threads In First And Second Portions Of Memory.....	53
3. All Claims: Stuttard Does Not Disclose or Render Obvious The Claimed Arbiter For Ground 2	59
4. Claims 3, 8: Williams Does Not Disclose A “Second Memory Device” .	59
5. Claim 7: Stuttard Does Not Disclose Or Render Obvious A Texture Processing Engine.....	60
6. No Motivation to Combine Or Reasonable Expectation of Success.....	60
C. Ground 3: Williams, Whittaker, and the Knowledge of a POSITA Does Not Render Claims 1-9 of the '053 Patent Obvious.....	65
1. All Claims: Ground 3 Does Not Disclose Or Render Obvious Storing Vertex Or Pixel Command Threads	66
2. All Claims: Williams’ Does Not Disclose Or Render Obvious The Claimed Arbiter	68
3. Claims 2, 5-9: Williams Does Not Disclose Or Render Obvious The Claimed Command Processing Engines.....	70
4. Claims 3, 8: Williams Does Not Disclose A “Second Memory Device” .	72
5. Claim 6: Williams Does Not Disclose Or Render Obvious Arithmetic Logic Unit.....	72

TABLE OF CONTENTS
(Continued)

	<u>Page</u>
6. Claim 7: Williams Does Not Disclose Or Render Obvious A Texture Processing Engine.....	72
7. No Motivation to Combine Or Reasonable Expectation of Success.....	73
X. CONCLUSION.....	76

I, John Hart, being of legal age, hereby declare, affirm, and state the following:

I. INTRODUCTION

1. I have been retained by ATI Technologies UCL (“ATI” or “Patent Owner”), as an independent expert in this proceeding before the Patent Trial and Appeal Board (“PTAB” or “Board”). I understand that the Board has instituted an Inter Partes Review (“IPR”) proceeding of U.S. Patent No. 7,742,053 (the “’053 Patent”) (Ex. 1001) in IPR2023-00564 requested by Petitioner Realtek.

2. I am providing this testimony for consideration by the Patent Trial and Appeal Board in this proceeding. Many of my opinions are also responsive to the Declaration of Massoud Pedram (“Pedram Decl.”) (Ex. 1004), which I understand was submitted in support of Realtek’s Petition for *Inter Partes* Review of the ’053 Patent. I have personal knowledge of the facts and opinions set forth in this declaration and all the opinions and conclusions found in this declaration are my own.

3. I am being compensated at my standard hourly rate for my services. I have no personal or financial stake or interest in the outcome of the present proceedings, and my compensation is not dependent in any way upon the outcome of this proceeding.

II. QUALIFICATIONS AND EXPERIENCE

4. I have been a professor of computer science, focusing on computer graphics, since 1992. I am currently a tenured Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign, a department consistently ranked in the top-5 by US News and World Report. I am also Director of the Office for Professional Education of the University of Illinois at Urbana-Champaign.

5. I received a Bachelor's of Science degree in Computer Science from Aurora University in 1987, a Master's of Science in Electrical Engineering and Computer Science from the University of Illinois at Chicago in 1989, and a Ph.D. in Electrical Engineering and Computer Science from the University of Illinois at Chicago in 1991.

6. I worked as an intern in 1998 at the IBM TJ Watson Research Center in Hawthorne, New York, and in 1999 at AT&T Pixel Machines in Holmdel, New Jersey. I worked as a postdoc in 1991-1992 at the Electronic Visualization Laboratory at the University of Illinois at Chicago with funding from the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. I worked as a visiting researcher in the summer of 2007 for the Graphics Research Group of Adobe Systems in Seattle, Washington.

7. From 1997-2000, I designed, simulated and patented graphics hardware for Silicon Reality and the Evans & Sutherland Computer Corp. I have also consulted for visual effects companies (e.g., Kleiser-Walczak, Blue Sky | VIFX), defense contractors (e.g., SAIC, Pratt & Whitney) and medical imaging companies (e.g., Intrinsic Medical Imaging).

8. My research in computer graphics has been supported by Adobe, DARPA, Intel, Nokia, NVIDIA, NSF and Microsoft. I have published over 75 reviewed papers on computer graphics, including papers in the most rigorous and prestigious venues, including the SIGGRAPH Annual Conference Proceedings, the ACM Transactions on Graphics, and the IEEE Transactions on Visualization and Computer Graphics. From 2002-2008, I was the longest-serving Editor-in-Chief of ACM Transactions on Graphics, the top journal in the field of computer graphics.

9. I was an Executive Producer for the ACM SIGGRAPH documentary “The Story of Computer Graphics” in 1999. I served from 1996-2000 on the ACM SIGGRAPH Executive Committee. I served as the founding computer graphics area editor of the ACM Books series, and oversaw the publication of “The VR Book: Human Centered Design for Virtual Reality.”

10. I co-authored the book “Real-Time Shading” in 2002 on the implementation of procedural shaders on graphics processors became fully user programmable GPU’s. I contributed two chapters to the book “Modeling and

Texturing: A Procedural Approach” in 2002, including a chapter on procedural texturing using rasterization and texturing operations.

11. I am being compensated for my time working on this matter at my usual rate of \$575 per hour. My compensation does not depend on the outcome of this litigation.

12. I believe I meet the criteria for a POSITA as laid out in Investigation No. 337-TA-1318, Ex. 1010 (ITC Markman Order), 9, and any other appropriate definition of a POSITA for this patent.

13. Additional details about my employment history, fields of expertise, and publications are further described in my curriculum vitae, which is attached as Exhibit A to this report.

III. MATERIALS CONSIDERED

14. In forming the opinions set forth herein, I have considered and relied upon my education, knowledge in the relevant field, and my experience. I have also reviewed and considered the materials cited herein, including the following materials:

Exhibit No.	Exhibit Title
Ex. 1001	U.S. Patent 7,742,053 to Lefebvre et al.
Ex. 1002	Prosecution History of U.S. Patent 7,742,053
Ex. 1004	Declaration of Massoud Pedram, Ph.D.
Ex. 1005	PCT Pub. No. WO00/62182A2 to Stuttard et al.
Ex. 1006	PCT Pub. No. WO00/63770A1 to Williams et al.

Exhibit No.	Exhibit Title
Ex. 1007	U.S. Patent 5,968,167 to Whittaker et al.
Ex. 1008	Curriculum Vitae of Massoud Pedram, Ph.D.
Ex. 1010	<i>Certain Graphics Systems, Components Thereof, and Digital Televisions Containing The Same</i> , Inv. No. 337-TA-1318 (Order No. 31, Oct. 18, 2022)
Ex. 1011	<i>LG Elecs., Inc. v. ATI Techs. ULC</i> , IPR2015-00325, Paper 62 (Final Written Decision) (PTAB Apr. 14, 2016)
Ex. 1012	U.S. Patent 7,116,332 to Boyd et al.
Ex. 1013	U.S. Patent 7,038,685 to Lindholm et al.
Ex. 1014	H. Selzer, Dynamic Load Balancing Within a High Performance Graphics System, Proceedings of the Sixth Eurographics Conference on Advances in Computer Graphics Hardware: Rendering, Visualization and Rasterization Hardware (Sept. 1, 1991)
Ex. 2002	<i>Certain Graphics Systems, Components Thereof, and Digital Televisions Containing The Same</i> , Inv. No. 337-TA-1318, Final Initial Determination (July 7, 2023)
Ex. 2003	James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. 1990. Computer Graphics: Principles And Practice (2nd ed.). Addison-Wesley Longman Publishing Co., Inc
Ex. 2004	Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). Real-Time Rendering (3rd ed.). A K Peters/CRC Press.
Ex. 2005	<i>Certain Graphics Systems, Components Thereof, and Digital Televisions Containing The Same</i> , Inv. No. 337-TA-1318, Hearing Transcript, Day 4 (March 16, 2023)
Ex. 2006	Edwin Earl Catmull. 1974. "A Subdivision Algorithm For Computer Display of Curved Surfaces." Ph.D. Dissertation. The University of Utah
Ex. 2007	Difference Between Buffering https://techdifferences.com/difference-between-buffering-and-caching-in-os.html
Ex. 2008	<i>Certain Graphics Systems, Components Thereof, and Digital Televisions Containing The Same</i> , Inv. No. 337-TA-1318, Deposition of Dr. Pedram (January 04, 2023)

Exhibit No.	Exhibit Title
Ex. 2010	Deposition Transcript of Massoud Pedram, Ph.D. (Jan. 5 and 8, 2024).
Ex. 2011	Frank D. Luna. 2003. Introduction to 3D Game Programming with DirectX 9.0. Wordware Publishing Inc.
Ex. 2012	Sumair Ahmed. 2009. “OpenGL – Lighting, Material, Shading and Texture Mapping.” IIT Bombay.

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

IV. LEGAL STANDARDS

16. In preparing and expressing my opinions and considering the subject matter of the '053 Patent, I am relying on certain basic legal principles that counsel have explained to me. These principles are discussed below.

A. Claim Construction

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

18. I further understand that an inventor can provide special definitions for specific claim term(s) within the patent. However, any special definition for a claim term must be set forth in the specification with reasonable clarity and precision.

B. Obviousness

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

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

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

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

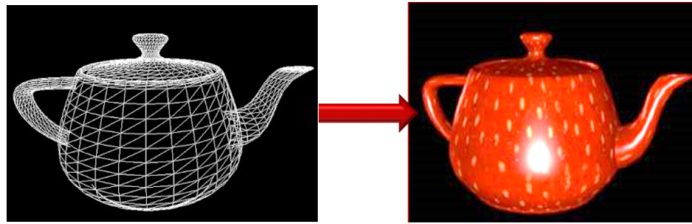
23. I have also been informed and understand that when considering the obviousness of a patent claim, one should consider whether a reason or motivation

existed for combining the elements of the references in the manner claimed, and that the prior art must create a reasonable expectation of success in producing the claimed subject matter.

V. RELEVANT TECHNICAL BACKGROUND AT THE TIME OF THE '053 PATENT

24. To the extent it may be helpful to the Board in understanding the context of the '053 Patent and in framing my opinions, I offer the following summary of pertinent technical information that would have been available to a person of ordinary skill in the art as of August 24, 2001, which I understand is the earliest priority date for the '053 Patent and is thus the lens through which the subject matter is to be evaluated.

25. Graphics processing involves manipulating and rendering 3D graphical objects onto a 2D display screen. Ex. 2004 (Real-Time Rendering) at 11. Specifically, 3D objects (typically represented by a collection of “primitives,” geometrical shapes such as triangles) are converted into a 2D image (typically comprising a collection of pixels) as shown below. Ex. 2003 (Foley) at 3, 11-15, 17-20; Ex. 2004, 11-13. The 2D image represents a depiction of the 3D objects, rendered from the perspective of a given viewer, and shaded to convey the object's surface material illuminated from a given light source. Ex. 2003. 210-222, 722-741; Ex. 2004, 21-27.



26. The conversion of a 3D shape consisting of a mesh of triangles into a photorealistic 2D image consisting of an array of pixels happens in a pipeline of stages. Ex. 2003, 806-809, 866-871; Ex. 2004, 12. The first stage is often referred to as “vertex processing” and computes, for the 3D position in space of each vertex (the point in space corresponding to the junction where neighboring triangles meet), a new 2D position of that vertex in the space of the screen or output image from the perspective of the viewer. Ex. 2003, 806-809, 866-871; Ex. 2004 (Real-Time Rendering) at 14-21.

27. The next stage, “scan conversion,” begins the “rasterization” process by converting the 2D triangle described by the three 2D vertices output from the vertex processing stage, into the pixels on the screen that lie within that triangle. Ex. 2003, 14, 806-809. This the scan conversion stage emits all the screen-space pixels that are covered by the triangle. Ex. 2003, 806-809; Ex. 2004, 21-22. The final “pixel processing” stage computes the colors of each of these pixels/fragments, and applies textures. Ex. 2003, 868-872; *see generally* Ex. 2006 (Catmull); Ex. 2004, 22-23.

VI. U.S. PATENT NO. 7,742,053

28. The '053 Patent is directed to graphics processing systems, and specifically towards the processing of multiple command threads in parallel. The '053 Patent disclosed novel methods for processing multiple threads across one or more hardware processing units with the specialized parallel processing architecture of a graphics processor.

29. Prior art systems were rigid systems that required providing command threads sequentially. *Id.* at 1:31-48. Another consequence of this architecture was the lack of support for an unlimited number of dependent texture fetches. *Id.* at 2:2-6. Prior to the '053 Patent, conventional systems consisted of two separate command processing engines: a pixel command processing engine that only ran pixel commands and a vertex command processing engine that only ran vertex commands. The vertex command processing engine would first process vertices. The triangle primitives defined by these processed vertices would then be scan converted into the pixels that filled them in. These pixels would then be processed by the pixel processing engine to determine their final color. Since there were 10x to 100x more pixels than vertices, graphics processing could run into a load imbalance causing one of the command processing engines to wait while the other finished its processing.

30. The '053 Patent solves those problems with a novel architecture that uses a single “unified shader processor” (the “command processing engine”) capable

of executing both pixel commands and vertex commands, along with the necessary arbitration to determine how to efficiently process commands given that there were more pixel processing commands than vertex processing commands:

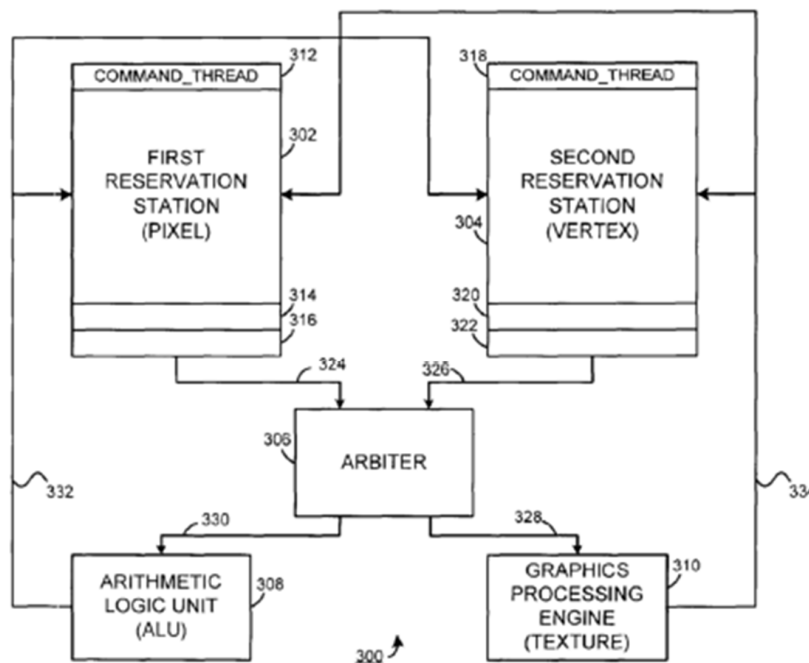


FIG. 4

Id. at Fig. 4.

31. This architecture further comprises a memory for storing the pixel and vertex command threads and an arbiter that selects one of those stored command threads for the unified shader based on the relative priorities of those threads. Ex. 1001 ('053 Patent), 2:45-48, 3:29-36. As a result, the arbiter 309 in Fig. 4 above could ensure that the unified shader processor was kept as busy as possible while it completed rendering of the current frame as quickly as possible, *e.g.*, either

processing the final pixels of the current frame, or by processing the vertices for triangles in the next frame. *Id.* at 3:16-20.

VII. DESCRIPTION OF ASSERTED ART

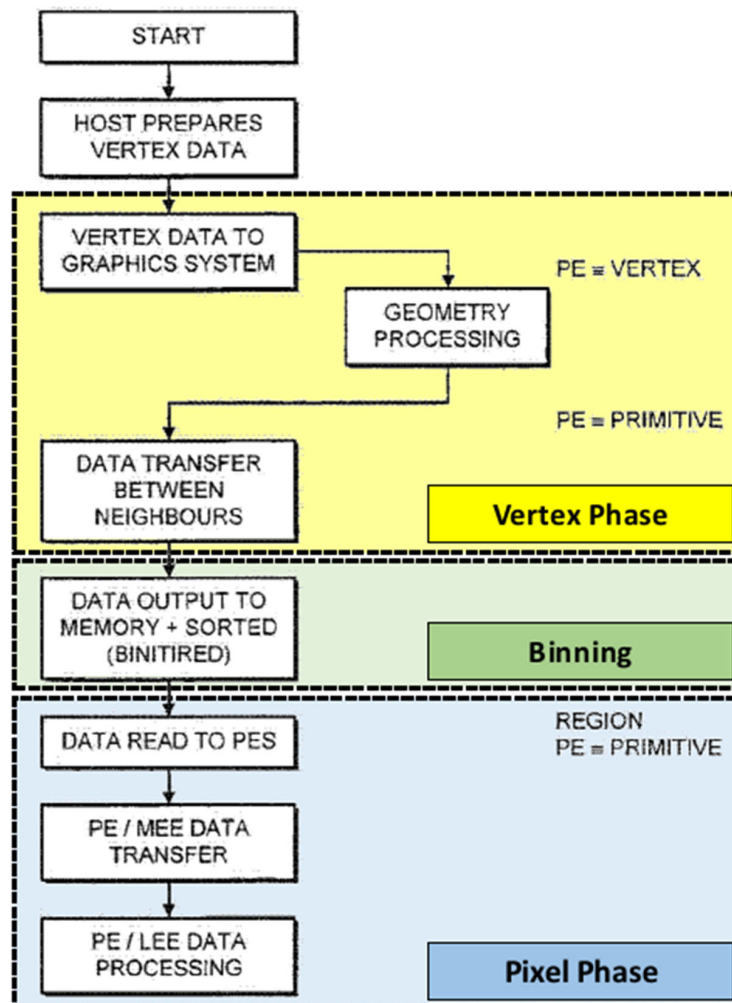
A. Stuttard (PCT Pub. No. WO00/62182A2)

32. Stuttard (Ex. 1005) is directed to a system which attempts to improve the use of a “single-instruction, multiple data” (“SIMD”) architecture by dividing the processing elements into processing blocks. Ex. 1005, 2:1-24. Specifically, Stuttard discloses “*a SIMD . . . array of processing elements, wherein the processing elements are operably divided into a plurality of processing blocks, the processing blocks being operable to process respective groups of data items.*” *Id.* at 2:19-23 (emphasis added).

33. In Stuttard, however, all the processing blocks of the SIMD operate on “*a common received instruction.*” *Id.* at 2:25-34; *see also id.* at 11:33-35 (“[T]he *same instruction stream* is supplied to all of the processing blocks in the system.”). Thus, at any given time, all processing units within Stuttard are operating on the same instruction. This is distinct from the parallel approach in the ’053 patent, described above.

34. Also critical to Stuttard’s architecture is that it operates first exclusively on vertices in one phase, and only when that phase is fully complete, then operates exclusively on pixels in a later phase, with no overlap in the execution of these two

phases. As explained more fully below, when Stuttard is operating in one phase, it does not store the alleged command threads relating to the other phase in memory (in particular in the small, fast cache memory Dr. Pedram focuses on) and would have no reason to be stored there at that time.



Id. at Fig. 9 (annotated).

35. The architecture of Stuttard is illustrated and described in Fig. 1 reproduced below. As shown in Figure 1, Stuttard discloses that the system

interfaces with a host system (not shown) via the interface 2. The graphics system 3 includes an embedded processing unit (EPU) 8 “for controlling the overall function of the graphics processor and for interfacing with the host system.” *Id.* at 4:26-30. The graphics processing capability of Stuttard’s system is provided by a processing core 10. *Id.* at 7:25-27.

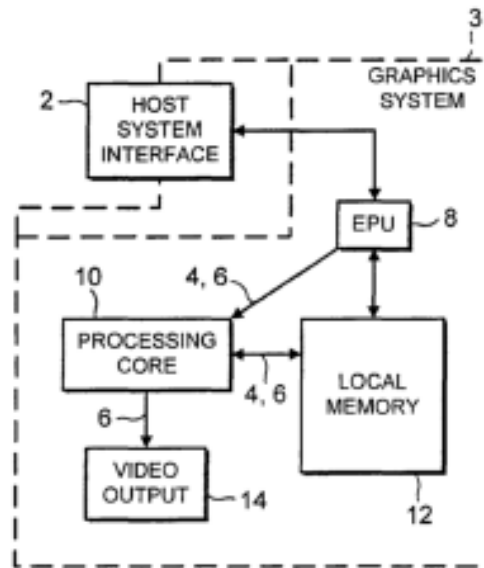


FIG. 1

Id. at Fig. 1.

36. The processing core includes processing blocks 106, which comprise an array 1061 of processor elements (PEs), a mathematical expression evaluator (MEE), and a binning unit. *Id.* at 8:15-9:3. To interface with and manage thread execution by the processing core, Stuttard describes the use of a thread manager 102,

an array controller 104, and a channel controller 108, which are illustrated and describe in connection with, for example, Figure 3 (reproduced below).

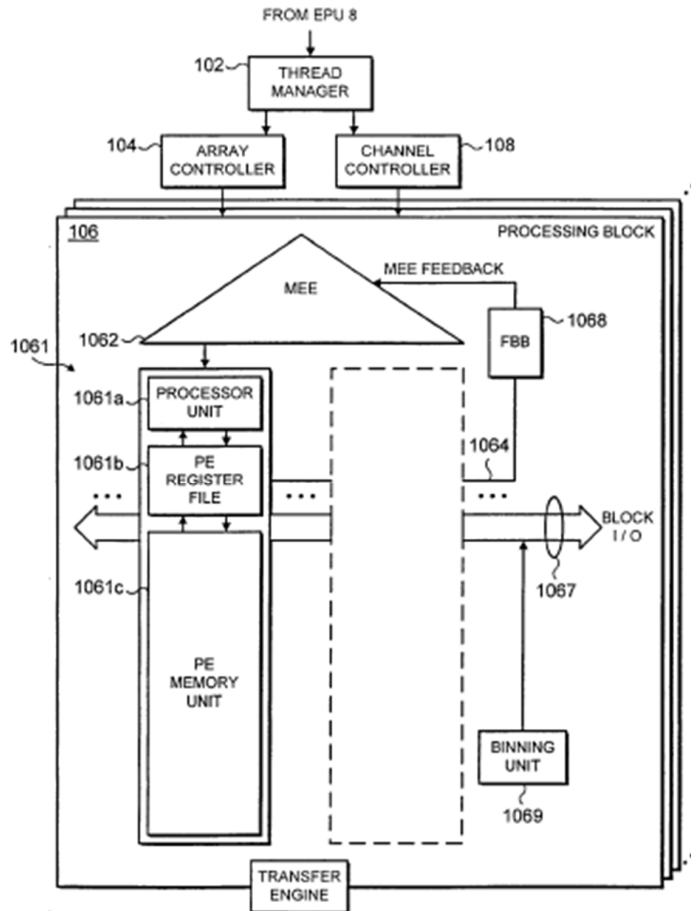


FIG. 3

Id. at Fig. 3. “The thread manager 102 is connected to receive control signals from the EPU 8. The control signals inform the thread manager as to when instructions are to be fetched and where the instructions are to be found. The thread manager 102 is connected to provide these instructions to the array controller 104 and to the channel controller 108. The array and channel controllers 104 and 108 are connected

to transfer control signals to the processing blocks 106 dependent upon the received instructions.” *Id.* at 8:17-27. Stuttard further discloses that “[e]ach processing block includes a binning unit 1069” and that “[t]he purpose of the binning unit 1069 is to sort primitive data by region, since the data is generally not provided by the host system in the correct order for region based processing.” *Id.* at 8:34-35; 14:30-35.

B. Williams (PCT Pub. No. WO00/63770A1)

37. Williams (Ex. 1006) discloses an apparatus and method for improving a graphics bus. It targets the efficiency of transmitting graphics commands from a CPU to a graphics subsystem, by distributing the transmission across several busses and reconstructing the original order of commands through the use of tags. *See* Ex. 1006, Abstract. Specifically, Williams identified the speed of graphics data transmitted from a CPU to a graphics subsystem is limited by the bandwidth of and congestion at the peripheral bus. *See id.* at 3:27-4:5. Williams’s disclosure is directed at “increase[ing] the throughput of graphics commands that can be transferred between the system bridge and a graphics subsystem.” *Id.* at 8:4-10.

38. Williams, in the non-regrouping embodiment, uses a buffer to temporarily store graphics commands. *Id.* at 9:10-12. In this embodiment, the graphics commands can be transferred from the system bridge over bus lines via a sorting unit. *Id.* at 9:8-10. The graphics commands are delivered to the graphics subsystem in the same order as they are transferred from the CPU. *Id.* at 23:14-19

(providing that graphics commands are transferred, “according to the tags, such that the multiple graphics commands regain the specific order (that is, their original order)).

C. Whittaker (U.S. Patent No. 5,968,167)

39. Whittaker (Ex. 1007) discloses a generic multi-threaded “data processing management system” to “process multimedia data.” Ex. 1007, Abstract. This “management system” consists of a “media control core” and “a number of data processing units.” *Id.* The “multimedia core identifies the data processing operation to be executed as well as the resources needed to execute that operation.” *Id.* The media control core “determines which operation has the highest priority” and passes that operation on “to one of the data processing units for execution.” *Id.*

VIII. CLAIM CONSTRUCTION

40. I understand certain terms of the '053 Patent have been previously construed by the Board and by the ALJ in the ITC Investigation. The ALJ construed certain terms in a claim construction order (Ex. 1010) during the pendency of the investigation. He further construed other terms still in dispute after the evidentiary hearing in his initial determination. Ex. 2002 (ITC Initial Determination), 32-43.

41. The table below summarizes the Board’s adopted construction of certain terms.

Claim Term	The Board’s Adopted Construction
“arbiter”	“any computer hardware, software, or combination thereof that receives and provides acommand thread” Ex. 1011 (IPR2015-00325 FWD), 7.
“command processing engine”	“as requiring each command processing engine be coupled to an arbiter and operable to receive and process a command thread selected by the arbiter” Ex. 1011, 12.

42. The table below summarizes the ALJ’s adopted construction of certain terms in his claim construction order.

Claim Term	ALJ’s Adopted Construction in Claim Construction Order
“arbiter,” “select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads,” and “relative priorities”	“Plain and ordinary meaning.” Ex. 1010 at 18-20.
“command threads”	“‘sequence of commands applicable to the corresponding element,’ with no requirement that they comprise state bits.” Ex. 1010 at 22-23.
“memory device”	“Respondents’ proposed construction of ‘memory device’ is rejected, which resolves the parties’ dispute, and the term is otherwise not construed.” Ex. 1010 at 23-24.
“coupled to”	“Respondents’ proposed construction of ‘coupled to’ is rejected, which resolves the parties’ dispute, and the term is otherwise not construed.” Ex. 1010 at 24-25.

43. Out of the above constructions, I understand Dr. Pedram has applied the narrower of the available constructions for terms that have more than one construction. Ex. 1004 (Pedram Decl.), ¶42.

44. The table below summarizes the ALJ’s adopted construction of certain terms in his initial determination, which was released after this petition was filed. I have applied the following constructions in my analysis.

Claim Term	ALJ’s Adopted Construction in Initial Determination
“at least one memory device...operative to store a plurality of pixel command threads and...a plurality of vertex command threads”	“[A]t least one memory device...operative to store a plurality of pixel command threads and...a plurality of vertex command threads” means operative to store a plurality of complete sequences of pixel commands and a plurality of complete sequences of vertex commands. Ex. 2002, 39. “[I]t is a correct claim construction that ‘at least one memory device comprising a first portion operative to store a plurality of pixel command threads and a second portion operative to store a plurality of vertex command threads’ means the memory device(s) are operable to store both pluralities at the same time.” Ex. 2002, 84.
“select a command thread”	“[S]elect a command thread does not mean selecting one and only one thread. Ex. 2002, 41.
“command processing engine”	“[A] person of ordinary skill would have understood a “command processing engine” could be implemented in software, hardware, or a mix of the two.” Ex. 2002, 43.

45. Additionally, my opinion with respect to the below terms is as follows. For all other claim terms, I applied their plain and ordinary meaning as it would be understood by a POSITA.

A. Storage of “command threads”

46. “Command thread” has been construed as “a sequence of commands applicable to the corresponding element.” Institution Decision (“ID”), 14-15. The “corresponding element” for a pixel command thread is a pixel, and for a vertex command thread is a vertex. Ex. 1001, 2:41-45.

47. All the challenged claims recite a memory device having a first portion that stores “a plurality of pixel command threads” and a second portion that stores “a plurality of vertex command threads.” This claim term refers to complete command threads, and subsequently requires the complete threads be stored by the memory device at any given moment in time, not merely a partial sequence of instructions that do not constitute a complete command thread. The claim language itself mandates this construction, by requiring the storage of and selection of “command threads,” and not of, *e.g.*, instructions of a thread. Ex. 1001, Claims 1, 5.

48. The specification is also consistent with this construction. In all instances, the disclosures describe the storage of a “command thread,” and not merely instructions. *See generally* Ex. 1001. For example, the specification

describes reservation stations as “any type of memory device capable of reserving and storing *command threads*.” *Id.*, 2:39-41 (emphasis added). Both Fig. 2 and Fig. 4 embodiments describe “command threads” as being stored. *Id.*, 2:63-3:36, 3:62-4:64, Figs. 3-4. Figs. 6 and 7, which illustrate flowcharts of exemplary methods also provides steps in which retrieving “a selected command thread.” *Id.*, 5:43-6:54, Figs. 6-7.

49. Additionally, complete command threads are described as being selected by the arbiter and returned to the memory device despite the execution of part of that command thread. The specification states that after a thread is selected by the arbiter, it is “submitted to the appropriate execution unit,” and “[u]pon the execution of *the associated command* of the command thread, the *thread* is thereupon returned to the station 302 or 304 at the same storage location with its status updated, once all possible sequential instructions have been executed.” *Id.*, 4:21-27 (emphasis added). Thus, the specification contemplates executing part of a command thread, but provides nonetheless that the entire thread is stored in memory device.

50. Thus, should be construed so that the claimed memory device must be operative to store a plurality of complete sequences of pixel commands and a plurality of complete sequences of vertex commands.

B. “at least one memory device . . . operative to store a plurality of pixel command threads and . . . operative to store a plurality of vertex command threads”

51. This claim limitation requires the storage of both pixel and vertex command threads at the same time. This is the plain language of the claims, which provides that the memory device is “operative to store a plurality of pixel command threads *and* a second portion operative to store a plurality of vertex command thread.” Ex. 1001, Claims 1, 5.

52. The '053 Patent specification uniformly describes the claimed memory device as storing both types of command threads at the same time. *Id.*, 2:65-3:3, 3:40-44 (“the sequencer logic 236 receives a first thread 242, and a second thread 244 from the buffer 234, such that the logic 236 may perform simultaneous, interleaved execution of the command threads”), 3:50-53 (“The arbiter selects the command thread by looking at the reservation stations, herein vertex and pixel reservation stations, and picking the first command thread ready to execute.”), 3:62-4:51, 5:10-16, 5:43-56, Fig. 2, Fig. 4, Fig. 5. For example, both pixel and vertex command threads are provided to reservation stations, “based on whether the command thread is a pixel command thread, such as thread 312, or a vertex command thread, such as thread 318.” *Id.*, 4:5-13. The arbiter then “selectively retrieves *either a pixel command thread*, such as command thread 316, *or a vertex*

command thread, such as command thread 322.” *Id.* In order to retrieve either type of command thread, both types must be stored at the same time. *See id.*

53. As additional rationale, both claim 1 and claim 5 require that the arbiter “select a command thread from either of the plurality of pixel command threads **and** the plurality of vertex command threads.” Ex. 1001, Claims 1, 5 (emphasis added). Yet if the only pixel command threads were present (and thus available) in the claimed memory device, the claimed arbiter would not be able to select from either of the pluralities, as required. The same is true if only vertex command threads were available. Thus, at a minimum, the claim requires the claimed memory device to store two pixel command threads and two vertex command threads, and thus four command threads, at the same time.

54. Therefore, the limitation “at least one memory device . . . operative to store a plurality of pixel command threads and . . . a plurality of vertex command threads” requires the storage of a plurality of pixel command threads and a plurality of vertex command threads at the same time.

C. “select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads”

55. Dr. Pedram argues that interpreting this claim element to require four command threads, two pixel and two vertex, to be stored in memory to select from renders the phrase “from either of” meaningless. Ex. 1004, ¶44. Yet it is Dr.

Pedram's interpretation which actually renders the phrase meaningless. "From either of" must be read in conjunction with "and," meaning that both types of command threads must be present. If the phrase "from either of" in the larger phrase "select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads" meant that only one type needed to be stored, it would, by the same argument, mean that only one type needed to be selected from.

D. "command processing engine"

56. The "command processing engines" claimed by claim 2 and 5 (and its dependent claims) must be able to process whichever thread presented to it by the arbiter, meaning that it is capable of processing both pixel and vertex command threads. A POSITA would understand this type of command processing engine to be a "unified shader." Dr. Pedram agreed that a unified shader includes "a command processing engine that can perform both kinds of vertex and pixel processing." Ex. 2005 (ITC Hearing Tr.), 916:2-6.

57. Claim 5 of the '053 Patent claims a "command processing engine" which is a unified shader. This is because the claim requires "a plurality of command processing engines, coupled to the arbiter, each operable to receive and process the command thread." Ex. 1001, Claim 5. Further, the claimed arbiter "is operable to select a command thread from either the plurality of pixel command threads *and* the

plurality of vertex command threads.” *Id.* (emphasis added). Thus, because the arbiter sends either a pixel command thread or a vertex command thread down to the command processing engine, that command processing engine has to be able to process whatever command thread is provided to it by the arbiter, so it has to be able to process both pixel command threads and vertex command threads. Claims 5-9 further explicitly require that the command processing engine be able to process whichever pixel or vertex command threads are selected by the arbiter.

58. Further, claim 2 also claims a “command processing engine, coupled to the arbiter, wherein the arbiter is further operable to provide the command thread to the command processing engine.” Ex. 1001, claim 2. Although claim 2 does not explicitly require the command processing engine to processing command thread, a POSITA would understand that the command processing engine must be capable of processing the command thread provided to it by the arbiter. This is because, as a POSITA would understand, an arbiter would not provide a command thread to a command processing engine which could not process it.

59. The ’053 Patent when describing the invention does not disclose specialized command processing engines, *i.e.*, command processing engines which process only pixel or only vertex command threads.

60. Therefore, “command processing engine” of both Claims 2 and 5 should be construed as being able to process both pixel and vertex command threads,

and not encompassing specialized engines that can only process one type of command thread.

IX. EVALUATION OF PETITIONER’S PROPOSED GROUNDS

A. Ground 1: Stuttard and the Knowledge of a POSITA Does Not Render Claims 1-9 of the ’053 Patent Obvious

61. In my opinion, claims 1-9 are not obvious in view of Stuttard and the knowledge of a POSITA. Stuttard does not disclose the claimed “at least one memory device” because the alleged memory device does not store both pixel and vertex command threads at the same time, a required limitation of all challenged claims. Nor is such a modification obvious in light of Stuttard. Stuttard discloses a *phased* system with a single SIMD, and its alleged memory device would never store pixel command threads during the phase in which it is exclusively processing vertices, nor vertex command threads during phase in which it is exclusively processing pixels. Further, even if the memory device is not required to store complete pixel and vertex command threads at the same time, Stuttard does not disclose the claimed “arbiter,” because Petitioner’s alleged arbiter cannot select from the pluralities of both types of command threads. This is because Stuttard’s phased operation would prohibit it from storing both types at the same time and prevent the requisite pool required for the arbiter to select from.

1. All Claims: Stuttard Does Not Disclose or Render Obvious the Claimed “at least one memory device”

62. Claims 1 and 5 require “at least one memory device comprising a first portion operative to store a plurality of pixel command threads and a second portion operative to store a plurality of vertex command threads.” Ex. 1001, Claims 1, 5. In my opinion, this claim limitation requires the storage of pixel command threads and vertex command threads in different portions of memory *at the same time*. See *supra* Section VIII.A.

63. Dr. Pedram identifies the cache unit 1024 as disclosing the claimed “at least one memory device” Ex. 1004, ¶59. However, even Dr. Pedram admits that “*Stuttard’s* FIFO cache unit 1024 does not explicitly disclose ‘a first portion operative to store a plurality of pixel command threads’ and ‘a second portion operative to store a plurality of vertex command thread.’” *Id.*, ¶62. Dr. Pedram relies on an alleged “obvious” modification to the memory device to store both types of command threads and to do so in separate “portions.” Ex. 1004, ¶¶62-67.

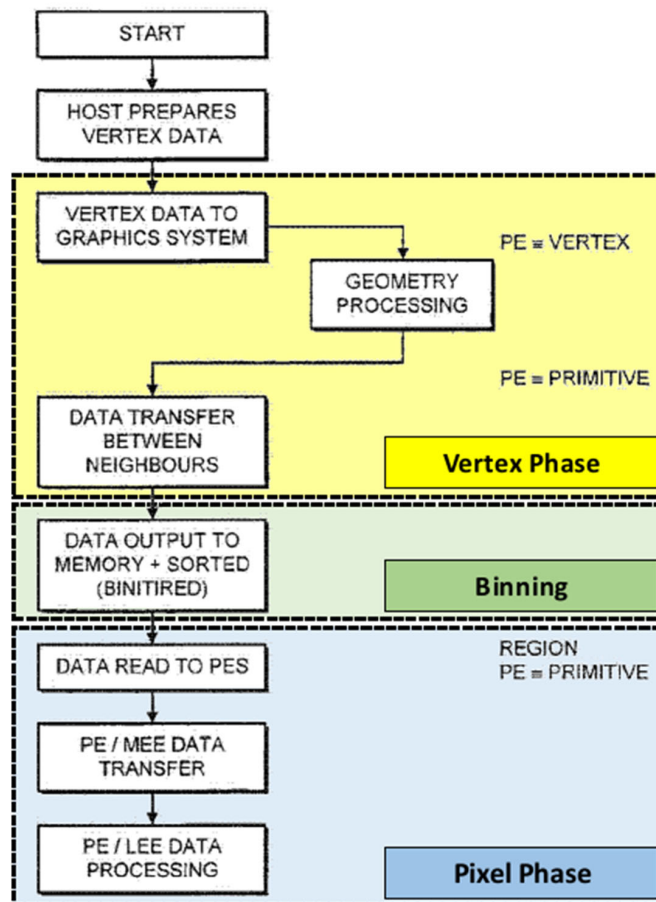
64. There is no motivation for Dr. Pedram’s proposed modification, for two reasons. *First*, Stuttard operates in a phased manner which requires completion of the processing of vertex instructions prior to pixel instructions, and so there is no reason to store vertex instructions and pixel instructions at the same time—much less the additional requirement dedicate one “portion” to pixel and another “portion”

to vertex. Such resource allocation would waste the valuable, limited resources of a small cache memory to store threads that are not capable of being executed part of the time. **Second**, Stuttard's system is a single SIMD, which only operates on one instruction at a time, as explained below in Section IX.A.1.b. So, there is no reason or even *ability* to modify Stuttard to be a non-phased system or to dedicate portions of memory to both vertex and pixel threads simultaneously. **Third**, there is no reason to store both command types in particular portions of memory. Dr. Pedram's stated rationale is that it is beneficial for the system to know where particular command thread types are located in memory so it can find the ones it wants faster. Ex. 1004, ¶66. But Stuttard's alleged arbiter, the thread manager (in conjunction with the thread processors) do not know or care about command thread type. Ex. 1005, 23:35-24:6. They only care about priority, and so there is no motivation to store command threads in particular portions according to type, because the entity that does the alleged "selecting" of command threads does not care about type.

a. Stuttard Operates In A Phased Manner

65. Figure 9 describes the phased graphics processing method of the system disclosed by Stuttard. Ex. 1005, 3:21-22 ("Figure 9 is a flowchart illustrating data processing using the system of Figures 1-8."). The method starts by processing vertices into primitives before undertaking a binning step. *Id.*, 20:25-21:26. Only after binning does Stuttard process pixels by region where each processing block

works on one region. *Id.*, 22:26-32. Before the binning step can occur, however, Stuttard provides that *all* vertices must be processed because they otherwise could not be sorted by region. *See id.*, 21:15-26. Dr. Pedram nor its expert does not even address the binning aspect of Stuttard, which is fatal to its arguments.



Id., Fig. 9 (annotated).

66. Figure 9 explains that the first step of graphics processing is that the host processes vertex data for the images to be processed and displayed by the graphics system. *Id.*, 20:25-30. The vertex data is “loaded into the PEs of the

graphics system so that each PE contains data for one vertex.” *Id.*, 20:32-33. Each PE processes the vertex data “to transform it from the host system reference space to the required screen space.” *Id.*, 21:3-5. The processed vertex data, now in the form of primitives, is output from the PEs to the binning units so that they can be sorted by region. *Id.*, 21:15-21. The sorting is required because data is not provided by the host system in the correct order for region-based processing. *Id.* Therefore, prior to binning, the vertex processing that happens is not region-based, *i.e.*, one region of the screen being assigned to one processing block.

67. The binning process mandates the processing of vertices and pixels in distinct *block parallel* phases. First, Stuttard explains that binning “must maintain primitive order between the geometry and rasterisation phases due to requirements of most host systems.” *Id.*, 41:12-24. Both geometry and rasterization phases are “block parallel,” meaning that the processing blocks run those phases in parallel, at the same time. *See id.* As further support, Stuttard provides a “mechanism for transferring data between any block to any of the bins and between any bin and any block,” during either of the two phases. *Id.* “This is implemented by creating multiple bin lists per region, one for every processing block 106 that is processing geometry data,” which “allows the geometry output phase to proceed in block parallel mode.” *Id.* Additionally, the pixel processing phase is also block parallel;

“[a] record is kept of how many primitives are written to each bin, so that regions can be sorted into similar size groups for block parallel rasterisation.” *Id.*, 42:1-10.

68. While each processing block has a binning unit, all binning units work collectively to complete the binning process. *See id.*, 41:5-10. This is demonstrated by the mechanism allowing for the transfer of data between any processing block to any of the binning units and between any binning unit and any processing block. *Id.*, 41:14-17. Further, a single channel controller controls all the binning units. *Id.*, 14:20-23, Fig. 3. It issues binning instructions which “relate to data transfers between PE memory and external memory where the data flows through the binning unit of each core block between the block I/O bus and a system bus to external memory.” *Id.*, 40:21-24.

69. After binning, several other processing steps must occur before the processing blocks can begin to process pixel instructions. First, the primitive information is read back into individual PEs, and “all of the PEs in one processing block contain data concerning respective primitives occurring in a single region.” *Id.*, 22:4-10. From this point, a given processing block operates on data associated with a single region of the display. *Id.* Next, each PE must transfer its data concerning its primitive to the MEE for scan conversion. *Id.*, 22:12-24. This determines whether a pixel appears within a primitive or outside of it, and each PE contains data about one pixel. *Id.* This step is necessary because while all the

primitives are determined, they may not show up on the screen because of viewing angles, lighting, or shading, etc. Ex. 2010, 234:22-7. Only after this scan conversion can the pixel data be processed for the pixels that occur within the primitive. *Id.*, 22:26-32.

70. In sum, the binning process means logically a processing block cannot proceed to pixel processing of a region until the system has determined which primitives are in that region. Therefore, Stuttard's system cannot determine which region each primitive belongs to until: (i) all the primitives for an image have been generated by the processing blocks by processing vertex data, and (ii) the binning process sorts the primitives by region. This is especially so because primitive data "is generally not provided by the host system in the correct order for region based processing." *Id.*, 21:18-21. Even after binning, the primitives must undergo scan conversion before any pixel processing can begin. Thus, Stuttard's phased system stands for more than the proposition that one must have a pixel to shade before shading a pixel.

71. Dr. Pedram's testimony that Stuttard may start processing pixels before vertices are completed is wrong, and completely divorced from any disclosures in Stuttard. *E.g.*, 2010, 236:11-237:24. While it is "possible," *id.*, 237:18, that some graphics processing systems may work in this way, that is not how Stuttard operates given its disclosures of phased operation, binitization, and the fact that region-based

processing occurs only after binitization for scan conversion and pixel processing. Thus, there is no circumstance where Stuttard would be able to divert vertex processing resources to pixel processing for one region prior to binitization given that Stuttard does not conduct region-based processing during the vertex phase.

72. There is also no circumstance would a POSITA understand that Stuttard's system would divert processing power to vertices for the next frame prior to completion of pixel processing for the prior frame. This is especially true given that processing of pixels takes more resources than the processing of vertices. Nor would a POSITA be motivated to modify Stuttard's memory device to store pixel instructions during the vertex processing phase (and vice versa) because it would be a waste of space and resources, and prevent the processing blocks from obtaining instructions relating to vertex processing as soon as possible.

73. Making such modifications would be antithetical to rendering frames in the shortest amount of time possible. Pixel processing is more resource intensive than vertex processing, and it does not make sense to divert resources to begin processing vertices of the next frame. Because GPUs are judged on their frame rates, which is how quickly a GPU can render a frame and display it on the screen, the GPU would want to finish pixel processing as fast as possible. No POSITA would be motivated to make modifications which would result in a slower frame rate.

b. Stuttard Executes One Instruction At A Time

74. Stuttard states its system is “a data processing apparatus comprising *a SIMD* (single instruction multiple data) array of processing elements, wherein the processing elements are operably divided into a plurality of processing blocks.” Ex. 1006, 2:17-23 (emphasis added). A SIMD array of processing elements is only capable of processing a single thread at any given moment in time.

75. Stuttard makes disclosures that its single SIMD array system is capable only of processing the same instruction across the entire array of processing blocks. First, Stuttard discloses that all its processing blocks operate on a “common received instruction.” *Id.*, 2:25-34. In another instance, Stuttard discloses that in conjunction with Figure 4, that the same thread (which Stuttard calls an “instruction stream”) “is supplied to all the processing blocks in the system.” *Id.*, 11:33-35.

76. Further, Stuttard’s system includes a thread manager, which controls the entire array of processing blocks. *Id.*, 8:16-24, 10:5-17, Fig. 3. When describing how the thread manager operates, Stuttard provides: “At a given time, only one thread is executing, and the scheduling of the time multiplexing between threads is determined by the dynamic conditions of the program execution.” *Id.*, 23:18-21. Stuttard further provides that the thread scheduler 1025, of which there is only one, “determine[s] which thread should be active at any one time.” *Id.*, 12:15-18. This is reiterated in a section describing the function of the thread scheduler in more

detail, entitled “Determining which thread should be active.” *Id.*, 23:34. The section explains that the thread scheduler will recalculate which thread should be active given certain circumstances. *Id.*, 23:34-25:17. All these disclosures are consistent with Stuttard having only a single active thread at one point in time.

77. The entire array of processing blocks within Stuttard’s system is controlled by the thread manager. The main purpose of the thread manager is to control which thread is currently active, and how to switch from one thread to another. Stuttard explains that the thread manager allows for faster switching in response to external conditions. *Id.*, 10:33-11:1 (“In addition, the system can have a faster response time to external events. Assigning particular threads to wait on different external events, so that when an event happens, it can be handled immediately.”), 25:5-9 (“If the previously active thread is yielding, the scheduler activates the highest priority READY thread, or the lowest priority thread if no thread is ready (since this will cause another immediate scheduling trigger).”). Stuttard’s detailed disclosure regarding the thread manager is focused on how to switch from one thread to another thread. *See generally id.*, 23:1-27:32.

78. Dr. Pedram identifies a disclosure describing thread processors which “control the issuance of core instructions from the thread manager so as to maintain processing of simultaneously active program threads, so that each the processing blocks 106 can be active for as much time as possible.” *Id.*, 11:29-33. This is

directed at how the thread processors within the thread manager operate, and not the processing blocks. The purpose of the thread processors “are to allow high level flow control to be performed for a thread (such as looping and conditional branches), and to assemble instructions to be issued to the array controller 104 and channel controller 108.” *Id.*, 26:3-7. Thus, the thread processors prepare threads for execution and ensure that other threads are ready to be executed when the active thread stalls. As further evidence that this disclosure does not disclose that the processing blocks can execute multiple tasks or even threads at one time, the following sentence indicates that “the same instruction stream is provided to all of the processing blocks in the system.” *Id.*, 11:33-35.

79. Stuttard provides there are several benefits in grouping the array of processing elements into processing blocks which are unrelated to executing multiple threads at one time. These benefits explain the division of the SIMD array of processing elements into processing blocks. First, Stuttard emphasizes that processing blocks operate on different data, not different instructions. *Id.*, Abstract (“wherein the processing elements are operably divided into a plurality of processing blocks, the processing blocks being operable to process respective groups of *data* items”). This is consistent with Stuttard’s SIMD disclosure, which provides that the same instruction operates on multiple data items.

80. Accordingly, Stuttard provides the processing blocks can operate on different regions of the display, *i.e.*, different data groups. Stuttard's system includes binning units whose purpose "is to sort primitive data by region, since the data is generally not provided by the host system in the correct order for *region based processing*." *Id.*, 14:32-35 (emphasis added). During this process, the binning units within each processing block work collectively to retrieve primitive data from local memory, and sorts the primitives by region. *Id.*, 21:15-21. After sorting, the primitive data "is read back into the individual PEs," with "all of the PEs in one processing block contain[ing] data concerning respective primitives occurring in a single region." *Id.*, 22:4-10. "From this point, a given processing block operates on data associated with a single region of the display." *Id.* Thus, dividing the array of processing elements into processing blocks allows for more organization in processing different regions of the display.

81. Dividing the array of processing elements into processing blocks also allows for redundancy. As recognized by Stuttard, one concern with a large array of processing elements is that it "can result in a large die size for the manufacture of the device in a silicon device." *Id.*, 9:22-28. Accordingly, it is preferable to have redundant processing elements such that if one die area is faulty, another can be used in its place. *Id.* Dividing the array of processing elements into processing blocks allows for such redundancy.

82. Further, the Institution Decision misreads claims 103-105, which Dr. Pedram does not rely on. Claim 103 does not teach that each processing block accesses a segment of memory appropriate for the tasks in the PEs in that processing block are performing. First, Stuttard distinguishes between instructions and data, and describes that the memory unit within PEs are used for storing *data*, not instructions. *Id.*, 9:11-20 (“Each processing element (PE) 1061 includes a processor unit 1061a for carrying out the instructions received from the array controller, a PE memory unit 1061c for storing data for use by the processor unit 1061a . . .”).

83. Second, Claim 103’s SIMD array of PEs is further divided into processing blocks that process respective groups of *data* items. This is commensurate with Stuttard’s SIMD design because respective groups of data items refers to the data that the instructions act on, and do not refer to the instructions themselves. The SIMD design means that the same instruction are being performed on the different groups of data items. Meanwhile, the local memory unit is for storing *data items*, which is the data that instructions and threads act on. Stuttard distinguishes data from instructions by describing data as “graphics data” or “video data.” *Id.*, 4:5-9. Similarly, the segment register stores address information for those data items; it does not store instructions or threads. *Id.* (describing a “PE register file . . . for temporarily storing data that is being processed by the processor unit 1061a”).

84. Claims 104 and 105 simply build on claim 103 by providing additional information on how to locate *data items*, not instructions or threads, from the local memory unit. For example, claim 104 describes that the SIMD array of PEs receive one instruction (“an instruction item”) which includes address information referred to in claim 103. *See id.*, Claim 14 (distinguishing between “data items” and “instruction items”). Claim 105 simply provides that the processing element can add the segment information to the address information for a complete address for locating the data items referred to in claim 103. Thus, these claims also do not show that Stuttard can execute more than one thread at a time.

c. Stuttard Does Not Care About Command Thread Type

85. Dr. Pedram’s argument is Stuttard’s system would benefit from knowing where in memory commands of specific types are, because it would be “faster and more convenient to store and retrieve different types of command threads.” Ex. 1004, ¶63. But Stuttard does not disclose that the alleged arbiter (the scheduling module 1025, along with the thread processors 1026), performs that selection on the basis of command thread type. Ex. 1005, 23:35-24:6 (thread scheduler selects threads on the basis of priority). Therefore, a POSITA would not go through the trouble of modifying the memory of Stuttard to place particular command types in particular first and second memory locations. Doing so would

create overhead and inefficiencies in memory storage and access, and slow down the process by which the arbiter, which is agnostic to command type, access the data in memory.

d. The Claims Require More Than Mere Possibility, Happenstance, Or Capability

86. The claims of the '053 patent require that the memory device be “operative to” store pixel command threads and vertex command threads in first and second portions of memory. Ex. 1001, claim 1. And the arbiter must be “operable to select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads.” *Id.*

87. I understand “operative to” or “operable to” to be synonymous with “configured to.” This means meaning that the system is configured to actually perform a function, not merely that it is capable of doing so. That is, mere probability or happenstance is insufficient. Dr. Pedram has not shown an actual configuration to store the required command threads and select from them.

2. All Claims: Stuttard Does Not Disclose or Render Obvious Storing Complete Command Threads

88. Claims 1 and 5 of the '053 Patent require a memory device storing “command threads,” and an arbiter selecting “command threads.” Ex. 1001, Claims 1, 5. As explained above in Section VIII.A, the storage of partial threads is

insufficient to disclose the invention, even if they are stored over time, because the arbiter must select “command threads” from the memory device.

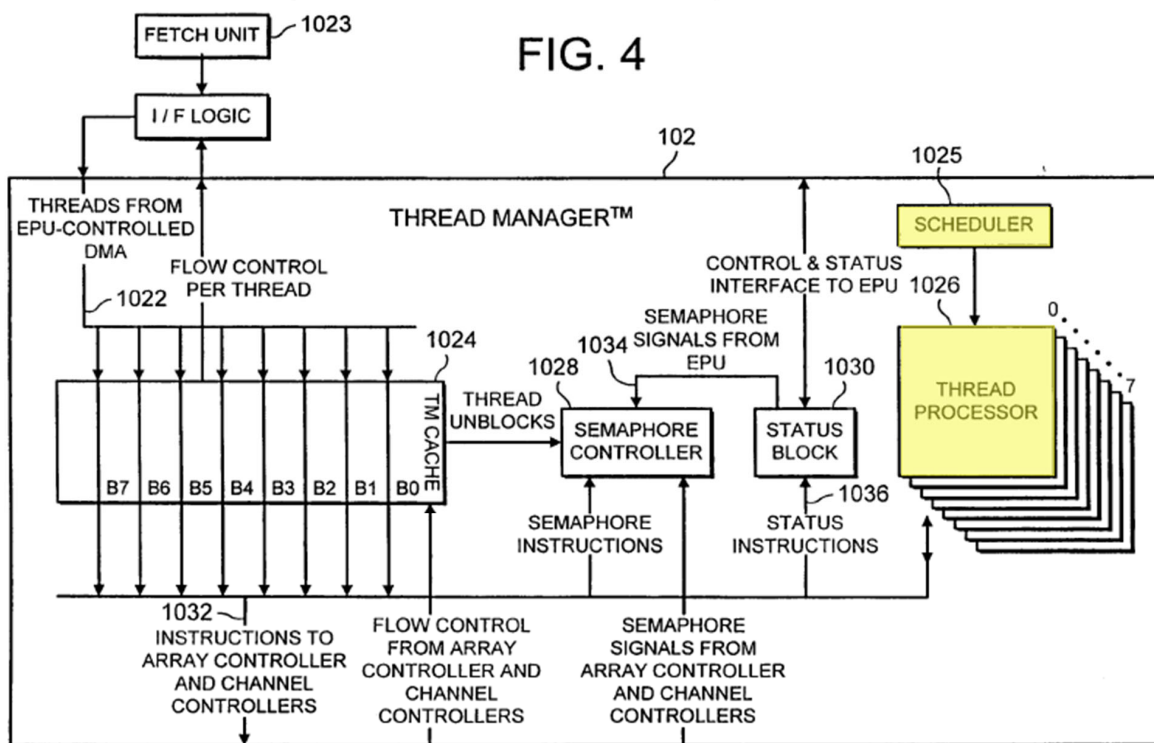
89. Dr. Pedram points to cache unit 1024 in Stuttard as the claimed “memory device.” Ex. 1004, ¶59. Stuttard discloses that cache unit stores instructions, *i.e.*, commands, not entire command threads. *See, e.g.*, Ex. 1005, 11:3-5 (“The thread manager 102 is shown in more detail in Figure 4, and comprises a cache memory unit 1024 for storing instructions fetched for each thread”), 11:12-14, 23:3-6, 23:13-16. Further, Stuttard discloses that instructions, not complete threads, are fetched by thread manager from host system and sent to cache unit for storage. “The thread manager 102 is connected to receive control signals from the EPU 8. The control signals inform the thread manager as to when *instructions* are to be fetched and where the *instructions* are to be found. The thread manager 102 is connected to provide these *instructions* to the array controller 104 and to the channel controller 108.” *Id.*, 8:17-24 (emphasis added). All these disclosures provide that instructions are fetched, and not complete threads. There is no disclosure that all the instructions for a thread are fetched and subsequently stored at the same time.

90. Additionally, the addition of the APA or any knowledge of a POSITA does not resolve this deficiency, nor has Dr. Pedram opined so. Dr. Pedram argues only that the APA teaches storing different types of command threads separately. Ex. 1004, ¶¶65-66.

3. All Claims: Stuttard Does Not Disclose Or Render Obvious An Arbiter That Selects A Command Thread From Either Of The Plurality Of Pixel Command Threads And The Plurality Of Vertex Command Threads

91. Claims 1 and 5 require “an arbiter, coupled to the at least one memory device, operable to select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads.” Ex. 1001, Claims 1, 5. At a minimum, the claim language itself requires the arbiter select from a pool of two pixel command threads *and* two vertex command threads.

92. Dr. Pedram opines that “thread scheduler 1025 and thread processors 1026” collectively correspond to the claimed “arbiter.” Ex. 1004, ¶68.



Ex. 1005, Fig. 4 (annotated).

93. Dr. Pedram’s opinion is deficient because it makes no allegation how the arbiter is “operable to select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads *based on relative priorities of the plurality of pixel command threads and the plurality of vertex command threads.*” Ex. 1001, Claims 1, 5 (emphasis added). The alleged arbiter cannot because, as explained above in Section IX.A.1, there is no point in time when Stuttard’s cache unit stores both pluralities of pixel command threads and vertex command threads, given Stuttard’s phased operation and SIMD design. Stuttard’s cache memory stores only vertex instructions during vertex processing, and pixel instructions during pixel processing. *See* Section IX.A.1.a, *supra*. Given Stuttard’s express disclosure of a phased system that executes a single vertex instruction during the vertex processing phase, and a single pixel instruction during pixel processing phase, there is never a chance for the alleged arbiter to select between at least two pixel command threads and at least two vertex command threads. *Id.*

94. Further, the alleged admitted prior art (“APA”), Figure 1 of the ‘053 patent, cannot supply the missing disclosure because that structure is fundamentally inconsistent with Stuttard’s phased operation and SIMD architecture that operates on one instruction at a time in a particular phase. Also, the APA never discloses the

type of command threads that are stored, only that generic command threads are stored. Ex. 1001, 1:31-64. It is not inherent from the disclosure that the command thread related to texture fetch is pixel, and the other command thread is vertex, or vice versa. *Id.*

95. The APA system uses a FIFO – first in first out – for each of the two buffers the arbiter has access to. Ex. 1001, 1:35-36, Fig. 1. Thus, the arbiter cannot *select* among the required four items – at least two vertex commands and at least two pixel commands; it only has access to the first item coming out of the buffer and does not select among commands of the same type (*i.e.*, the required “pluralities” component of the claim limitation). Even if Dr. Pedram is correct that the arbiter does not need access to both types of commands simultaneously, it must be able to “select” from a “plurality” of at least one type, which cannot happen using a FIFO. Further, upon selection, the arbiter provides the command thread to another buffer, and not a command processing engine as required. Ex. 1001, 1:49-57, Fig. 1. Nor does the APA disclose that the arbiter is coupled to a command processing engine. *Id.*, 1:58-64.

96. Dr. Pedram has not explained how Stuttard’s architecture would be modified to fit the architecture of the APA, or vice versa.

4. Claim 7: Stuttard Does Not Disclose Or Render Obvious A Texture Processing Engine

97. Dr. Pedram opines that texture processing engines are inherent due to Stuttard's disclosure of performing "lighting and shading." Ex. 1004, ¶84 ("A person of ordinary skill would have recognized that lighting and shading operations *necessarily require* a texture processing engine.") (emphasis added). In my opinion, a POSITA would not have recognized that lighting and shading operations "necessarily require" a texture processing engine, nor does Dr. Pedram provide any support for this assertion. Some lighting and shading processes only require vertex processing, and thus do not require a texture processing engine. This means that texture processing engines are not inherent to performing lighting and shading.

98. Further, a texture processing engine is more complex than simply processing lighting and shading. Texture processing engines apply textures to certain portions of an image. Processing lighting and shading does not require applying texture. Several publications around the time of the '053 Patent show that texturing was not required to process lighting and shading. *See, e.g.*, Ex. 2011 (Luna), 11 (discussing disabling a texture and code for rendering an object without texture), 5 (Figure 4.2), 8 (Figure 5.7), 4-10 (showing sample shading and lighting applications, and output screens without texture). In fact, literature well past 2003 also shows that texture operations were not a necessary step for lighting and shading,

and were in fact separate. See Ex. 2012 (Ahmed), 4-5, Fig. 2. Therefore, POSITAs would understand that lighting and shading operations are separate from texture mapping and thus do not necessarily require a texture processing engine

B. Ground 2: Stuttard, Williams, and the Knowledge of a POSITA Does Not Render Claims 1-9 of the '053 Patent Obvious

99. In my opinion, claims 1-9 are not obvious in view of Stuttard, Williams, and the knowledge of a POSITA.

1. All Claims: Williams Does Not Disclose Or Render Obvious “Command Threads”

100. Williams does not resolve the deficiencies in Stuttard described above, namely the failure to store complete threads and pixel and vertex command threads at the same time and in a first and second portion of memory, because Williams does not even disclose commands or command threads claimed by the '053 Patent. Williams' graphics commands are not pixel or vertex commands, but higher level API commands, such as OpenGL commands. In addition, even if Williams' graphics commands were lower level pixel/vertex commands (which they are not), Williams does not disclose storing complete command threads.

101. Nor has Dr. Pedram shown that Ground 2 is “operable to” store the required command threads and select from them, which requires that the combined system is configured to do so. See Section IX.A.1.d, *supra*.

a. Williams' Graphics Commands Are Not Command Threads

102. Williams does not store the “commands” of the claimed “command thread.” A “command thread” is a “sequence of commands applicable to the corresponding element.” Ex. 1004, ¶47. The '053 Patent specification provides that a pixel command thread is relative to processing of pixel elements and a vertex command thread is relative to processing of vertex elements. Ex. 1001, 2:41-45.

103. Williams' graphics commands instead are “a graphics application program interface (API) command, or other read or write type commands” or “any type of machine code command that graphics subsystem 102 understands (for example, a read or write command).” Ex. 1006, 6:24-7:2. Williams provides an examples of graphics commands as being commands written in OpenGL language or Direct3D. *Id.* Both OpenGL and Direct3D are examples of an application programming interface (API). In the graphics processing context, an API provides a way for software on a host system to communicate with a graphics processor. Table 2 provides examples of Williams' graphics commands. *Id.*, 19:4-21:9.

<u>BUS</u>	<u>Pseudo-Code of Graphics Command</u>
bus1	gl Finish() ;
bus1	glRenderMode (GL_RENDER) ;
bus1	glViewport (0, 0, 0x258, 0x258) ;
.	.
.	.
.	.
bus2	glTexImage2D (GL_TEXTURE_2D, 0, 0x3, 0x40, 0x40, 0, GL_RGBA, GL_UNSIGNED_SHORT, 0x107ab000) ;
bus2	glTexImage 2D (GL_TEXTURE_2D, 0x1, 0x3, 0x20, 0x20, 0, GL_RGBA, GL_UNSIGNED_SHORT, 0x107ab000) ;
bus2	glTexImage2D (GL_TEXTURE_2D, 0x2, 0x3, 0x10, 0x10, 0, GL_RGBA, GL_UNSIGNED_SHORT, 0x107ab000) ;
bus2	glTexImage2D (GL_TEXTURE_2D, 0x3, 0x3, 0x8, 0x8, 0, GL_RGBA, GL_UNSIGNED_SHORT, 0x107ab000) ;

104. *Id.*, Table 2 (excerpt). Therefore, Williams’ graphics commands differ from the command threads of the ’053 Patent. The API commands or OpenGL commands disclosed by Williams do not disclose vertex and pixel command threads, because they are higher level and could be implemented in myriad ways when they are received and processed. Depending on how the system was configured, or graphics commands may or may not result in the storage of vertex command threads and pixel command threads. There is not enough information in Williams to say that they would be, nor has Petitioner provided any analysis on this. It is dependent on unstated implementation details on how those commands would be carried out.

b. Williams Does Not Store Complete Command Threads

105. Even if the graphics commands were the lower level commands of a command thread, Williams does not rectify Stuttard’s deficiency of not storing

complete threads because it also does not disclose storing complete “command threads.” A “command thread” is a “sequence of commands applicable to the corresponding element,” where the “corresponding element” refers either to vertex data or pixel data. Ex. 1004, ¶47; Ex. 1001, 2:41-45. Thus, there must be some relationship between multiple instructions that are sequenced together, and they must be applicable to particular vertex data or pixel data, to constitute a thread. The challenged claims require storing a complete command thread, not just parts (*e.g.*, individual commands) that over time would constitute a command thread. *See* Section VIII.A, *supra*.

106. Nothing in Williams describes storing such a sequence, *i.e.*, a command thread as a whole. Williams only stores individual graphics commands, and does not even mention the word “thread” or any equivalent concept. *See generally* Ex. 1006. Dr. Pedram identifies the system memory and/or the input buffer of Williams’s Fig. 4 embodiment as disclosing the claimed “at least one memory device . . . limitation. Ex. 1004, ¶96.

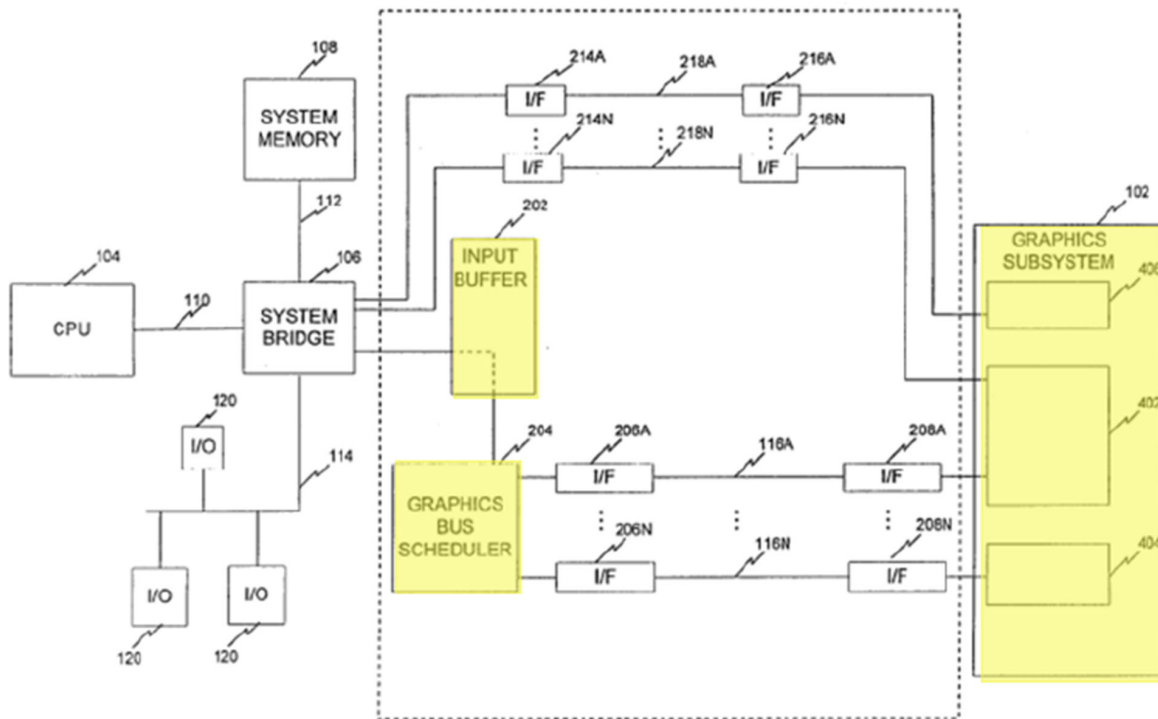


FIG. 4
NON-REGROUPING EMBODIMENT

Ex. 1006, Fig. 4. Williams explains that individual “commands” are put into the input buffer 202, without explaining any relationship between multiple commands such that they could even hypothetically together constitute a command thread. Ex. 1005, 20:14-26. Then “GBS 204 assigns/sorts graphics commands according to their type” and puts them on the appropriate bus. *Id.* Specifically, the “graphics commands are not reordered,” which at least means they are not grouped into a “thread.” *Id.*

107. Further, the storage devices (output buffer) in the Figure 6 embodiment Dr. Pedram relies on in the alternative, also stores commands and not command

threads. Ex. 1004, ¶100 (Ex. 1005, 23:1-21). This embodiment also describes transferring “commands” one by one across certain busses. *Id.*

108. The Institution Decision found that accumulation of commands in display list 704 discloses complete command threads. ID, 43-44 (citing Ex. 1006, 24:16-17). However, as explained above, a POSITA would understand that the claims require storage of a complete command thread, not an over-time storage of the parts of a command thread. The accumulation of various commands, without some disclosure that they together constitute a command thread cannot suffice to disclose the claimed memory device limitation which stores complete threads.

109. Further, display list 704 cannot be the claimed memory device because it is part of graphics pipeline 700, which is downstream from the input buffer (alleged memory device) and GBS (alleged arbiter). *Id.*, 23:23-34, 24:5-9, Fig. 7. The claimed arbiter must select from the pool of command threads that are stored within the memory device, and the alleged arbiter (GBS) cannot do so if the alleged memory device (display list 704) does not receive commands until after the alleged arbiter has transferred them, which is the case here.

110. Dr. Pedram also fails prove the more narrower requirement that the memory device must store at least two pixel command threads and vertex command threads. As described above in Section VIII.B, the memory device must store a minimum of two pixel and two vertex command threads in order for the arbiter to

be able to select from that minimum pool as required by the claims. Dr. Pedram does not even address this requirement with disclosures from Williams. Therefore, Williams cannot resolve this deficiency of Stuttard.

2. All Claims: Williams Does Not Disclose Or Render Obvious Storing Vertex and Pixel Command Threads In First And Second Portions Of Memory

111. Williams provides no indication of the claimed “first portion” and a “second portion” of memory, each portion respectively storing pixel and vertex command threads. It simply discloses a buffer for temporary storage of commands. Ex. 1005, 11:20-24 (“Input buffer 202 . . . temporarily stores graphics commands that are to be transferred from system bridge 106 to graphics subsystem 102.”), 16:28-30, 17:26-27. Dr. Pedram concedes this point and instead argues it would have been obvious to dedicate these first and second portions within memory “for efficiency of command fetching and transfer, which is Williams’ objective.” Ex. 1004, ¶99. Dr. Pedram’s alleged motivation for its proposed modification does not benefit Williams or Stuttard and would not be carried out by a POSITA.

a. There Is No Motivation To Modify The Memory In Williams

112. Dr. Pedram argues that “it was well-known that commands stored in consecutive addresses can be *read/written with fewer instructions* than the non-consecutive alternative because reading non-consecutive addresses involves address jumps.” Ex. 1004, ¶99. Those hypothetical benefits have no relevance to Williams.

In Williams, input buffer receives graphics commands and “stores the graphics commands such that their original order (i.e., the order in which the graphics commands were transferred from system bridge 106 to input buffer 202) is known.” Ex. 1006, 11:22-24. Although this quotation comes from the section regarding the “regrouping” embodiment, the earlier steps regarding how the Graphis Bus Scheduler receives commands is the same in the embodiments.

113. The GBS then receives commands one by one from the input buffer, “sorts the graphics commands according to the type of command,” and sends those commands to specific downstream components. Ex. 1006, 18:14-26. Williams’ system does not require “address jumps” because the commands are stored in input buffer in their original order, and retrieved one by one from input buffer by GBS in their original order. Ex. 1006, 8:13-14 (“graphics commands are transferred . . . in a specific order”), 8:25-9:1. Thus, GBS does not read memory addresses to obtain the commands. There is no benefit on the “read” end, when the commands are coming out of the input buffer, to storing particular command types in particular locations of memory, because the commands simply come out of the buffer as they went in.

114. The same is true on the “write” end; Stuttard uses a fast, low-overhead method of simply putting the commands into memory without any indication that particular locations are selected or remembered. Dr. Pedram apparently argues that

somehow it would require *fewer* instructions if the commands were required to be stored in particular locations. That is nonsensical: such a change strictly adds overhead and complexity, rather than reducing it.

115. Moreover, there is no benefit to the GBS to store particular command types in particular locations. The GBS does not care where the commands are. The way it functions is by determining what command type it received from the input buffer, either via the command information itself or via a tag or other metadata attached to the command. Ex. 1006, 21:13-22:2. Then it puts the command on the bus line for that command type. *E.g., id.*, 19:4-17. There is no reason to add a pre-sorting requirement such that commands of particular types are stored in first and second memory portions. Thus, any hypothetical benefit of eliminating address jumps and using fewer read instructions has no application to Williams because there are no address jumps or read instructions, and so it does not provide a motivation to modify Williams as Dr. Pedram suggests.

116. Modifying Williams to require reading particular memory addresses in order to retrieve commands would substantially slow the system down. For example, in order to store graphics commands in separate portions of input buffer, an additional component akin to GBS would be needed to identify the type of graphics command each command is and then direct them to their specific portion of input buffer. Further, making the proposed change so that given types of

commands were in predetermined memory locations would also render redundant the GBS unit, whose job it is to sort the commands by type based on the randomly-ordered assortment that exit the input buffer. Having two components with essentially the same function would lessen the efficiency of Williams. Therefore, Dr. Pedram has failed to meet its burden to show that its proposed system is more “efficient” than the system Williams uses, or any reason a POSITA would make the proposed change.

117. Dr. Pedram’s alternative argument, regarding Figure 6 of Williams, fails for the same reasons. Ex. 1004, ¶100; Dr. Pedram identifies Williams’ input buffer and output buffer of Fig. 6 (reproduced below) as disclosing the “at least one memory device” claim limitation. *Id.* Dr. Pedram asserts that the input buffer stores pixel commands and the output buffer stores vertex commands, thus meeting the claim requirements. *Id.* However, similar to deficiencies with display list 704 described above, output buffer cannot be the claimed memory device because this does not satisfy the requirement that the claimed arbiter must select from the claimed memory device. Simply put, the claimed arbiter must select *from* the pool of command threads stored by the memory device(s). But Dr. Pedram has not explained how its alleged arbiter for Ground 2 (Stuttard’s thread scheduler and thread processors together) or Ground 3 (Williams’ GBS), selects *from* output buffer. Without such explanation, based on how output buffer is situated in

Williams, output buffer *receives* the transferred commands, which occurs after Williams' GBS's purported selection. *Id.*, 23:16-21. Dr. Pedram has provided no indication the same would not be true with respect to both Grounds 2 and 3.

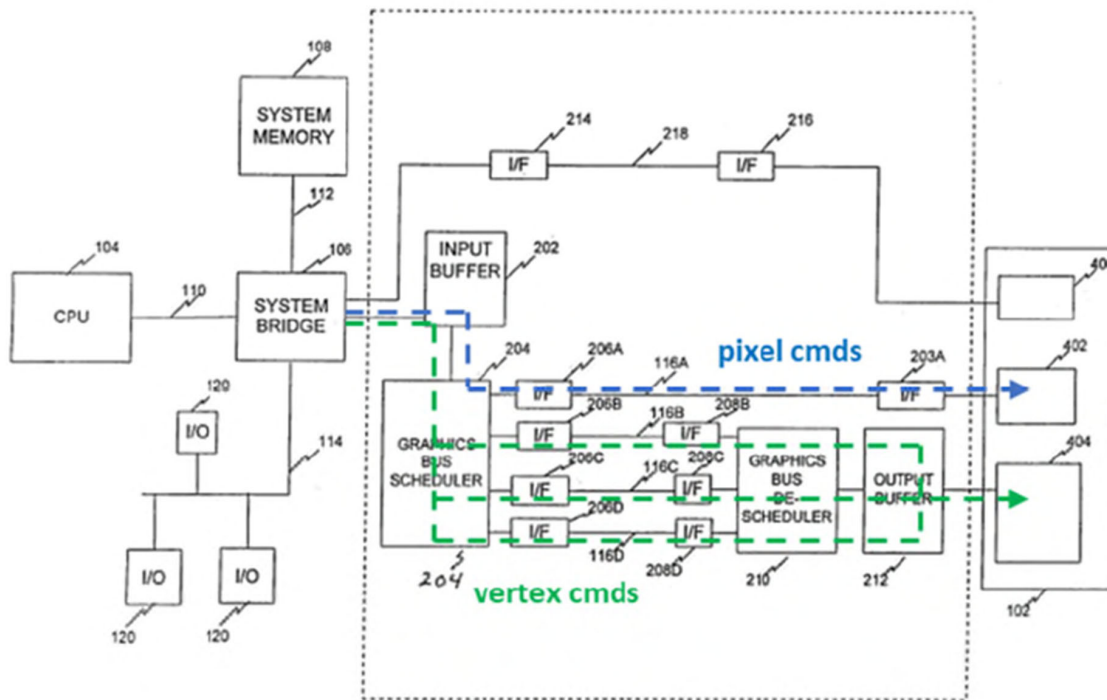


FIG. 6

Ex. 1006, Fig. 6 (annotated by Dr. Pedram); Ex. 1004, ¶100. Additionally, this argument suffers from all the same arguments above, *e.g.*, this disclosure merely stores commands and not command threads. Further, there is no indication that they are stored “at the same time,” as some relate to input, before the buses, and some relate to output, after the buses. *Id.* Finally, as indicated by Fig. 6 above, GBS is not “coupled to” output buffer where there are several bus lines and a graphics bus de-scheduler in-between the two.

**b. There Is No Motivation To Include The Modified
Version Of Williams's Memory In Stuttard**

118. For the reasons discussed above in Section IX.B.2.a, there is no reason to include the modified hypothetical memory of Williams, in which vertex and pixel command threads are allegedly simultaneously stored in a first and second portion of memory. Such a memory results in only wasted space and overhead processing, with zero benefit to the design of Stuttard's arbiter, which at no point cares what command type is being processed.

119. The Institution Decision also considered a theory in which the GBS and the type-specific bus lines of Williams were included in the combination. According to Dr. Pedram, Stuttard takes a flexible approach, its arbiter is not concerned with command type, and instead only concerned with command priority, in which any processing element can process anything. *See* Ex. 1004, ¶69. Williams takes a rigid approach in which a dedicated module (the GBS) identifies commands by type, loads them onto dedicated bus lines, sends them to dedicated processing modules that only process exactly that command type, and forgoes flexibility for certainty and predictability. *E.g.*, Ex. 1006, 19:4-7, 22:3-7. There is no benefit to adding a GBS to identify command type, and dedicated-command-type bus lines of Williams, into Stuttard.

**3. All Claims: Stuttard Does Not Disclose or Render Obvious
The Claimed Arbiter For Ground 2**

120. Dr. Pedram relies on the alleged arbiter from Stuttard for this ground. For the same reasons explained above, this ground does not disclose the claimed arbiter which selects from a pool of two pixel command threads and two vertex command threads. *See* Section IX.A.3, *supra*.

121. Moreover, Dr. Pedram fails to disclose how the same, unmodified arbiter in Stuttard would work when the memory changes as proposed. For that reason alone, the combination fails.

4. Claims 3, 8: Williams Does Not Disclose A “Second Memory Device”

122. For Ground 2, Dr. Pedram relies on Stuttard for claims 3 and 8. Ex. 1004, ¶109. He also identifies “output buffer” from Williams, opining that it is one of the two required “separate memory devices.” *Id.*, ¶110.

123. Williams explains that “GBS 204 transfers pixel (e.g., texture) related graphics commands across bus 116A directly to an appropriate functional component of graphics subsystem 102, and transfers geometry related graphics commands across busses 116B-116D to GBD 210. The geometry related graphics commands accepted by GBD 210 do not necessarily have the specific order that the commands had within input buffer 202. GBD 210 transfers the accepted multiple graphics commands into output buffer 212, according to the tags, such that the

multiple graphics commands regain the specific order (that is, their original order) within output buffer 212.” Ex. 1006, 23:11-19.

124. The output buffer cannot be one of the claimed memory devices. As explained above in Section IX.B.2.a, the alleged arbiter (GBS) does not *select from* the graphics commands stored in output buffer, as claimed. Claims 1 and 5 both require the “arbiter” is “operable to select a command thread *from* either of the plurality of pixel command threads and the plurality of vertex command threads” that are stored in the “at least one memory device. Ex. 1001, Claims 1, 5. However, Stuttard provides that output buffer is where graphics commands are end up after they have been sorted/sent by GBS. Ex. 1006, 23:11-19. Nor is GBS “coupled to” output buffer, as required by Claims 1 and 5, given that there are several bus lines and a graphics bus de-scheduler in-between GBS and output buffer. Ex. 1006, Fig. 5.

5. Claim 7: Stuttard Does Not Disclose Or Render Obvious A Texture Processing Engine

125. Ground 2 does not disclose or render obvious these claims for the same reasons explain in Section IX.A.4.

6. No Motivation to Combine Or Reasonable Expectation of Success

126. Even if Williams taught or rendered obvious the claimed memory device, which it does not as explained above, a POSITA would not be motivated to

combine Williams's system memory, input buffer, or output buffer with, or as a replacement for, Stuttard's cache unit 1024, as Dr. Pedram alleges. Dr. Pedram's motivation to combine argument rests on the two references being both directed to graphics processing, without taking into consideration the nuances of each. *See* Ex. 1004, ¶102 (proving that both references "have overlapping components, such as a CPU in a host system and a graphics system," etc.).

127. Dr. Pedram does not identify any specific components of Williams that it would add to or replace any of Stuttard's components. The combination relies mostly on Stuttard, with additions from Williams to the "at least one memory device" limitation of claims 1, 3, and 8. Ex. 1004, ¶¶101-102. However, the functionality of Williams's system memory, input buffer, and output buffer is already redundant of Stuttard's local memory and system memory devices. A POSITA would not replace Stuttard's cache unit with William's system memory or input/output buffers where the functionality of these devices is redundant of functionality already found elsewhere in Stuttard's system. *Id.* For example, Williams' input buffer, which is used "as an interface between system bridge 106 and GBS 204." Ex. 1005, 18:21-22. This functionality is redundant of the functionality already present in Stuttard's EPU and/or local memory, which also acts as a bridge between the host system and the graphics system and has storage for command threads. Ex. 1005, 7:1-9, 11:12-14, 23:1-16 ("storing instructions fetched

. . . from local external memory 103 or from the EPU 8 by the fetch unit 1023”), Fig.

2.

128. Further, a POSITA would understand that replacing Stuttard’s high speed cache with Williams’ lower speed input/output buffer would render Stuttard inoperable for its intended purpose of enabling high speed instruction execution. *See generally, e.g., Ex. 2007 (Difference Between Buffer)*. This is because buffers used by Williams are used to match the speed of sending and receiving data, and are less concerned about providing the fastest access to data. *Id.* Meanwhile, the cache used by Stuttard is used to store data which needs fast and repeated access. *Id.*

129. Moreover, even if Williams stored a plurality of vertex/pixel command threads in first and second portions, a person of ordinary skill in the art would not be motivated to modify Stuttard’s cache unit 1024 to incorporate this structure. As previously discussed, Stuttard discloses a phased, SIMD, structure, which is only capable of executing a single instruction at a time. As Stuttard’s system is incapable of executing pixel threads until it completes all vertex processing, much less at the same time as executing vertex threads, storing both of a plurality of pixel threads and vertex threads in separate first and second portions of the instruction cache 1024 would waste valuable storage resources to store threads that are incapable of being executed. *Id.* As a result of this inefficient use of a caching strategy, the modified version of Stuttard would encounter a greater number of cache misses, making

Stuttard's overall system performance worse. *Id.* Stuttard's alleged arbiter does not select on the basis of command type, only priority, meaning that modifying cache unit to store in separate portions based on command type is meaningless. Ex. 1005, 23:35-24:6. Modifying the cache to store by command type when it is not needed would increase the overhead and decrease performance as there is potential wasted space if the system doesn't have as many of either command types as expected. The modification has the downside of being less flexible.

130. Dr. Pedram argues a POSITA would be motivated to combine Stuttard and Williams because both share objective of increasing the speed of graphics processing and that those teachings are complementary. Ex. 1004, ¶103. However, the objective of Williams is not to increase the speed of graphics processing, but to increase the bandwidth of transferring data from a host system to the graphics subsystem. Ex. 1006, 1:5-7 ("The present invention relates to . . . increasing the bandwidth to a graphics subsystem."). In contrast, the objective of Stuttard is to increase processing efficiency of large numbers of small data item groups. Ex. 1005, 2:1-13. A POSITA would understand that increasing processing efficiency is a different objective than increasing the speed of transferring items to the graphics processor. Even if a POSITA attempted to address the goal of faster processing speed with faster bandwidth, a POSITA would only be motivated to increase bandwidth if that bandwidth was a limiting factor to the processing speed. There is

no indication that bandwidth from the host system to the graphics systems is a limiting factor in Stuttard.

131. Dr. Pedram also argues that the issues with AGP buses identified by Williams and used by Stuttard's system as a reason for a motivation to combine. Ex. 1004, ¶104. However, there is no indication that Stuttard's system is limited by bandwidth of the AGP busses. A POSITA would not be motivated to increase bandwidth unless there was a need to do so. Further, Williams also acknowledges that AGP buses have "relatively high bandwidth." Ex. 1006, 7:19-23. Stuttard also discusses the benefits of AGP buses, stating they have "high bandwidth" allowing for "large texture databases to be held in the host system memory" and "provid[ing] a mechanism for mapping memory between a linear address space on the graphics system and a number of potentially scattered memory blocks in the host system memory." Ex. 1005, 5:18-28. Stuttard does not identify any issues with AGP busses as causing a bottleneck to its system. *Id.* Further implementing Williams' invention may increase costs to Stuttard's system, especially when compared to well-developed and easily obtainable off the shelf components such as Intel's AGP. Such an increase in costs would be useless if the extra bandwidth is not needed, as is the case here. Therefore, a POSITA would not be motivated to exchange the AGP bus of Stuttard's system with William's invention.

132. Dr. Pedram argues a POSITA would have a reasonable expectation of success of combining Stuttard and Williams because they share common components, *i.e.*, a host system with a CPU and a graphics subsystem, where the host system provides data and graphics commands to the graphics subsystem. Ex. 1004, ¶102. However, these are generic reasons which are common to most graphics systems. Almost all graphics systems include a host system with a CPU and a graphics subsystem, and in such a configuration, the host system must have a channel of communication with the graphics subsystem to provide data and commands. Dr. Pedram's other reason, that both systems involve processing of conventional graphics objects including pixel and vertex commands, is also, by Dr. Pedram's own admission, conventional of graphics systems. The '053 Patent recognizes that "a typical graphics system . . . vertices and/or pixels[] are processed." Ex. 1001, 1:24-28. Based on Dr. Pedram's reasons, a POSITA would have a reasonable expectation of success in combining almost every reference on graphics systems, which is incomprehensible.

C. Ground 3: Williams, Whittaker, and the Knowledge of a POSITA Does Not Render Claims 1-9 of the '053 Patent Obvious

133. In my opinion, claims 1-9 are not obvious in view of Williams, Whittaker, and the knowledge of a POSITA. Neither reference discloses or renders obvious storing pixel and vertex command threads. Nor does Williams disclose or

render obvious the claimed arbiter, because its purported arbiter, at best, merely sorts graphics commands that it receives in a way that does not involve any selection. Williams also does not disclose or render obvious the “command processing engines” because Petitioner’s identified components do not process both pixel and vertex command threads. Nor is there sufficient motivation to combine or a reasonable expectation of success because Williams’ ordered graphics commands is incompatible with Whittaker’s reordering based on thread priority.

1. All Claims: Ground 3 Does Not Disclose Or Render Obvious Storing Vertex Or Pixel Command Threads

134. As explained above in Section IX.B.1, Williams does not disclose the storage of command threads. Dr. Pedram does not rely on Whittaker for this claim limitation. Ex. 1004, ¶132. Nor could it because Whittaker does not disclose pixel or vertex command threads. Whittaker’s discussion of processing threads according to available resources and priorities are not pixel or vertex command threads. Ex. 1007, 4:44-61. Specifically, Whittaker at 4:44-61 describes a cache bank for storing data, such as texture maps. The cache banks do not store threads. Rather, they increase *data* throughput. *See also id.*, 5:1-7 (describing “[m]ore cache banks” are “added to support further data processing units thereby increasing the effectiveness of the data throughput to the media control core”). Further, an

offhand reference to storing texture maps does not mean that Whittaker is focused on 3D graphics processing.

135. Whittaker is not focused on 3D graphics processing. *See generally* Ex. 1007. The description of its system contains no references to 3D graphics processing; it only mentions offhand the storage of texture maps. This reference would not enable a POSITA to understand that Whittaker addressed 3D graphics processing. Further, Williams provides nothing on how graphics processing would be implemented, not to mention whether vertex and pixel command threads would be used. Tellingly, Whittaker does not reference “vertices,” which are present in 3D graphics processing, at all. To the extent Whittaker discusses any type of processing in depth, it is focused on video processing, *i.e.*, a form of signal processing where input and output signals are video files or video streams, which is fundamentally different than graphics processing. For example, Whittaker explains how the resource and priority checking of media control core enables real time video data to be processed without large memory buffers. Ex. 1007, 4:15-25. Whittaker also describes how the digital to analog converter (DAC), used in video processing but not graphics processing, is incorporated into its system. *See generally id.*, 6:30-56. Whittaker mentions “pixels” only once and in the context of DAC video processing.

136. In contrast to 3D graphics processing described above in Section V, video processing relates to the processing of signals, in particular image processing, which often employs video filters and where the input and output signals are video files or video streams. Video processing techniques are used in television sets, VCRs, DVDs, video codecs, video players, video scalers and other devices. Video processing does not require using vertex command threads and pixel command threads because it does not involve transforming 3D objects into 2D screen space. Therefore, Whittaker does not disclose pixel command threads and vertex command threads, or render them obvious.

137. Separately, Dr. Pedram has not shown that Ground 3 is “operable to” store the required command threads and select from them, which requires that the combined system is configured to do so. *See* Section IX.A.1.d, *supra*.

2. All Claims: Williams’ Does Not Disclose Or Render Obvious The Claimed Arbiter

138. Independent claims 1 and 5 of the ’053 patent require “an arbiter, coupled to the at least one memory device, operable to *select* a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads.” Ex. 1001, Claims 1, 5 (emphasis added).

139. Dr. Pedram identifies GBS 202 is the claimed arbiter. Ex. 1004, ¶133. However, Williams’s GBS does not *select* graphics commands (or anything). Rather,

GBS “*receives* multiple graphics commands directly from system bridge 106, or from input buffer 202,” and “assigns each of the multiple graphics commands to one of the plurality of busses 116A-116N.” Ex. 1005, 18:26-28 (emphasis added). As explained above in Ground 2, the GBS merely receives the commands, one by one, coming out of the input buffer and sorts them; it does not select commands. A so-called arbiter which merely receives does not select. Additionally, the fact that GBS may recognize command types and sort based on that does not mean that it had a role in what arrived at its input terminals, as the sorting occurs after the command is received. And the sorting cannot be the claimed selecting. First, the claimed selecting must be done from the memories, as claimed. Second, there is no sense of “picking out” as the patentee stated was the meaning of “select.” Ex. 1002 (’053 File History), 107. The GBS simply puts the command on the bus corresponding to its type.

140. Even if that sorting could involve some sort of selection, Williams does not describe it. Williams does not describe any procedure that GBS follows with respect to sorting, including, *e.g.*, that several commands are provided at once and GBS first selects in any way one prior to sorting, only stating that it must be done. Dr. Pedram also fails to consider that Williams provides commands one by one. *Id.*, 308:10-17. Williams discloses this because it provides that the input buffer has the commands in order. Ex. 1006, 8:13-14 (“graphics commands are transferred . . . in

a specific order”), 8:25-9:1. And the order is not inherent to the commands themselves because GBS must tag them as they come out of the input buffer to preserve that order. *Id.*, 8:14-16. A POSITA would understand this is the most obvious way Williams would operate given that Williams does not disclose any details on how the GBS might “select” commands in the process of sorting them; more importantly, no other way is actually disclosed, which is insufficient to invalidate the claims.

141. Moreover, the claim requires selecting between at least two pixel command threads and two vertex command threads, by virtue of requiring a “plurality” of *each* type, *i.e.*, at least two of each type, which is certainly not taught by Williams or Whittaker.

3. Claims 2, 5-9: Williams Does Not Disclose Or Render Obvious The Claimed Command Processing Engines

142. Both claims 2 and 5 require a “command processing engine” that can process both pixel and vertex command threads. *See* Section VIII.C, *supra*. Despite the differences in claims 2 and 5, Dr. Pedram has not identified different evidence or provided any explanation as to why these claims are disclosed or obvious. As explained above, Williams in view of Whittaker does not disclose pixel command threads or vertex command threads. Therefore, Ground 3 does not disclose a

command processing engine to receive (and process) the selected command thread, which could be a pixel command thread or a vertex command thread.

143. Further, Dr. Pedram's identified components, "functional components" 402 and 404, are not the claimed "command processing engines." Ex. 1004, ¶140.

144. Functional components are type-specific, meaning that each functional component can only perform geometry operations or pixel operations, and not both. Williams discloses that functional components 402 and 404 "perform geometry and pixel operations, *respectively*," meaning that functional component 402 only performs (and thus receives) geometry operations, and functional component 404 only performs (and thus receives) pixel operations. Ex. 1006, 22:5-7 (emphasis added). Williams also discloses that the graphics commands can be "transferred to the appropriate component/stage of a graphics pipeline 700." Ex. 1006, 22:7-9; *see also id.*, 25:15-23. For example, graphics front end 702 separates geometry type graphics commands and pixel type graphics commands, and provides them to different components. *Id.*, 24:12-17. Geometry related commands are provided to evaluator 706 and pixel related commands are provided to pixel operations stage 716. *Id.* Additionally, even per-vertex operations and primitive assembly stage 708 and per-fragment operations stage 712 are type specific – one component is only dedicated to primitives/vertices and the other is dedicated to only fragments/pixels. Thus, only dedicated, type-specific components in Williams alleged to be the

claimed command processing engines. This is contrary to the claims, which require the command processing engines to process both pixel and vertex command threads.

145. Lastly, Dr. Pedram has made no attempt to demonstrate how functional components are coupled to the GBS 204, the alleged arbiter, as they must be per the claims. Dr. Pedram has also not explained how functional components receive both pixel and vertex commands from the GBS 204 (as required by claim 2).

4. Claims 3, 8: Williams Does Not Disclose A “Second Memory Device”

146. Ground 3 does not disclose or render obvious these claims for the same reasons explained in Sections IX.B.2, IX.B.4.

5. Claim 6: Williams Does Not Disclose Or Render Obvious Arithmetic Logic Unit

147. Dr. Pedram identifies disclosures from Williams mentioning, *e.g.*, geometry calculations, curve and surface geometry, processing geometric primitives, and operations on pixels. Ex. 1004, ¶148 (citing Ex. 1006, 2:26-30, 24:18-23, 24:27-25:2. But none of these reference the required arithmetic logic unit. Nor does Dr. Pedram explain how the alleged command processing engines, functional components, comprise an arithmetic logic unit processing engine.

6. Claim 7: Williams Does Not Disclose Or Render Obvious A Texture Processing Engine

148. Dr. Pedram solely identifies the disclosure that “graphics subsystem 102 ‘processes texture data.’” Ex. 1004, ¶149. This disclosure does not provide

where in graphics subsystem 102 texture data is processed, which is required by claim 7. Specifically, claim 7 requires that the plurality of command processing engines comprise a texture processing engine. Dr. Pedram's alleged command processing engines are functional components 402, 404. Ex. 1004, ¶149. As explained, functional components are not command processing engines. *See* Section IX.C.3, *supra*. But putting that aside, Dr. Pedram has made no showing that any alleged texture processing engine is part of either of the functional components.

7. No Motivation to Combine Or Reasonable Expectation of Success

149. A POSITA would not have been motivated to combine Williams and Whittaker or have a reasonable success in doing so. First, the two references are focused on different art. Williams is focused on 3D graphics processing whereas Whittaker is focused on video processing. Second, Williams and Whittaker disclose incompatible methods of optimization.

150. Dr. Pedram argues a POSITA would have a reasonable expectation of success in combining the two references because both are directed to the field of 3D graphics. Ex. 1004, ¶138. However, Whittaker is directed to video processing, not 3D graphics. As explained above in Section IX.C.1, Whittaker focuses much of its discussion on video and audio input data, which is not relevant to 3D graphics processing, and components such as DAC and VGA which are relevant to video

processing. *See, e.g.*, Ex. 1007, 4:15:25, 5:53-60, 6:13-56, Fig. 3. Whittaker explains that its MCC “will look to see whether [video input] data is available at the IO port and, if it is, will receive that data and send it” to memory storage in preparation for processing. *Id.*, 4:15:25.

151. Figure 3 is a block diagram of Whittaker’s system, and includes components relevant to video processing, namely video inputs and outputs, audio inputs and outputs, DAC core and pipeline, VGA (video graphics adaptor), and frame buffer interface. *Id.*, Fig. 3.

152. Whittaker makes only a single reference to storing texture maps in the context of 3D graphics processing. *Id.*, 4:48-50. The storage of texture maps discloses nothing regarding how Whittaker might hypothetically implement 3D graphics processing, and Whittaker makes no reference to the processing of pixel and/or vertex instructions.

153. Second, the thread priority used by Whittaker is incompatible with Williams’ ordered graphics commands. Williams targeted high-performance graphics through bus optimization, seeking to “increase[] the throughput of graphics commands transferred between a system bridge and a graphics subsystem.” Ex. 1006, Abstract. Key to Williams is maintaining the original order of the graphics commands, which may be jumbled when being shuttled over the system bridge to the graphics subsystem. Ex. 1005, Abstract, 9:2-7, 18:14-22, 23:1-10. To maintain

the original order of the graphics commands when they reach the graphics subsystem, Williams discloses either tagging the graphics commands so that they can be reordered, or sending graphics commands directly to specific functional components within the graphics subsystem. *Id.*

154. In contrast, Whittaker sought a cost-saving general multimedia processor by prioritizing which threads are being processed. Ex. 1007, Abstract, 2:4-6. Whittaker discloses a media control core which checks resource availability, and then from the operations for which all the resources are available, “determines which operation has the highest priority.” *Id.*, Abstract; 3:56-4:14. Thus, Whittaker describes a system that is concerned with looking at the priority of threads and being able to change the order in which threads are processed based on that priority. There is no regard to the order threads are delivered to the processor, as in Williams.

155. Adding an arbiter which selects based on priority (such as Whittaker’s media control core) to Williams contradicts how the graphics commands are stored in Williams’ input buffer – which is by order – and how they subsequently transferred to the graphics subsystem. In all Williams’ embodiments, there is no need to transfer commands based on priority because the order the graphics commands are transferred from the host is the order they are received by graphics subsystem. Therefore, the order of the graphics commands is important to Williams, and a POSITA would not add an arbiter which disturbs that order; a POSITA would

understand implementing priority-based reordering is not a benefit to Williams, it is a change that is in direct contradiction to the chosen approach.

156. Williams does not address factors such as resource availability and higher performance when selecting threads for execution, as Dr. Pedram argues. Williams does not discuss resource availability, nor does Williams select threads for execution.

157. Dr. Pedram also claims that both references share the objective of improving the efficiency of transferring data and commands to the graphics processing subsystem. Ex. 1004, ¶139. Whittaker discusses increasing efficiencies of data *processing*, not faster transfer graphics commands to a graphics subsystem. *See generally* Ex. 1007, 2:1-3:11.

X. CONCLUSION

158. In signing this declaration, I recognize that it will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place in the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

159. The statements and opinions in this declaration comprise and are based upon my review, analysis and understanding of the source materials that I have

reviewed as discussed herein, based on the disclosure and statements made within such source materials.

160. I reserve the right to supplement my opinions in the future to respond to any arguments that the Petitioner raises and to take into account new information as it becomes available to me.

161. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

January 17, 2024

Respectfully Submitted,



Dr. John Hart