

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

**PATENT OWNER'S RESPONSE
PURSUANT TO 37 C.F.R. § 42.107(A)**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	1
II. Graphics Processing.....	3
A. Vertex and Pixel Processing.....	3
B. Vertex and Pixel Command Threads	4
III. '053 PATENT	4
IV. EXEMPLARY CLAIM.....	6
V. CLAIM CONSTRUCTION	7
A. Storage of “command threads”	7
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”	8
C. “select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads”	10
D. “command processing engine”	11
VI. PETITIONER HAS FAILED TO ESTABLISH THE REQUIRED REASONABLE LIKELIHOOD OF SUCCESS FOR ANY GROUND	14
A. Ground 1: Stuttard In View of the Knowledge Of A POSITA.....	14
1. All Claims: Stuttard Does Not Disclose or Render Obvious a Memory Device That Stores a Plurality of Pixel Command Threads and Vertex Command Threads at the Same Time in First and Second Portions	20
a. Stuttard Operates in a Phased Manner.....	21
b. Stuttard Executes One Instruction at a Time.....	26
c. Stuttard Does Not Care About Command Thread Type	31
d. The Claims Require More Than Mere Possibility, Happenstance, or Capability.....	32
2. All Claims: Stuttard Does Not Disclose or Render Obvious Storing Complete Command Threads	33

3.	All Claims: Stuttard Does Not Disclose Or Render Obvious an Arbiter That Selects A Command Thread From The Plurality Of Pixel Command Threads Vertex Command Threads	36
4.	Claim 7: Stuttard Does Not Disclose Or Render Obvious a Texture Processing Engine.....	39
B.	Ground 2: Stuttard In View Of Williams and The Knowledge Of A POSITA	40
1.	All Claims: Williams Does Not Disclose or Render Obvious Storing Vertex and Pixel Command Threads in First And Second Portions of Memory	45
a.	There Is No Motivation to Modify the Memory in Williams.....	45
b.	There Is No Motivation to Include the Modified Version of Williams’s Memory In Stuttard.....	51
2.	All Claims: Williams Does Not Disclose or Render Obvious Storing “Command Threads”	52
a.	Williams’ Graphics Commands Are Not Pixel and Vertex Command Threads.....	53
b.	Williams Does Not Store Complete Command Threads	55
3.	All Claims: Stuttard Does Not Disclose or Render Obvious the Claimed Arbiter	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 In View Of Whittaker and the Knowledge Of A POSITA	62
1.	All Claims: Ground 3 Does Not Disclose or Render Obvious Storing Vertex or Pixel Command Threads	65

2.	All Claims: Williams’ Does Not Disclose or Render Obvious the Claimed Arbiter	66
3.	Claims 2, 5-9: Williams Does Not Disclose Or Render Obvious the Claimed Command Processing Engines	68
4.	Claims 3, 8: Williams Does Not Disclose A “Second Memory Device”	69
5.	Claim 6: Williams Does Not Disclose or Render Obvious Arithmetic Logic Unit.....	69
6.	Claim 7: Williams Does Not Disclose or Render Obvious a Texture Processing Engine.....	70
7.	No Motivation to Combine or Reasonable Expectation of Success	70
VII.	CONCLUSION.....	72

TABLE OF AUTHORITIES

	Page(s)
 Federal Cases	
<i>Aspex Eyewear, Inc. v. Marchon Eyewear, Inc.</i> , 672 F.3d 1335 (Fed. Cir. 2012)	33
<i>ATI Techs. ULC ATI Techs. ULC v. Iancu</i> , 920 F.3d 1362 (Fed. Cir. 2019)	12
<i>Gemalto S.A. v. HTC Corp.</i> , 754 F.3d 1364 (Fed. Cir. 2014)	13
<i>In re Magnum Oil Tools Int’l, Ltd.</i> , 829 F.3d 1364 (Fed. Cir. 2016)	35, 57
<i>Oren Techs., LLC v. Proppant Express Invs. LLC</i> , No. 2019-1778, 2021 U.S. App. LEXIS 21859 (Fed. Cir. July 23, 2021)	57
<i>Persion Pharms. LLC v. Alvogen Malta Operations LTD.</i> , 945 F.3d 1184 (Fed. Cir. 2019)	40
<i>Retractable Techs., Inc. v. Becton</i> , 653 F.3d 1296 (Fed. Cir. 2011)	13
<i>TQ Delta, LLC v. Commscope Holding Co.</i> , No. 2:21-CV-310-JRG, 2022 U.S. Dist. LEXIS 102577 (E.D. Tex. June 8, 2022).....	32
<i>Wasica Fin. GmbH v. Cont’l Auto. Sys.</i> , 853 F.3d 1272 (Fed. Cir. 2017)	70
 Regulations	
37 C.F.R. § 42.24(d)	73

TABLE OF EXHIBITS

Exhibit	Description
2009	Declaration of John Hart, Ph.D., Regarding U.S. Patent No. 7,742,053 (Jan. 17, 2024)
2010	Deposition Transcript of Massoud Pedram, Ph.D. (Jan. 8, 2024)
2011	Frank D. Luna. 2003. Introduction to 3D Game Programming with DirectX 9.0. Wordware Publishing Inc.
2012	Sumair Ahmed. 2009. “OpenGL – Lighting, Material, Shading and Texture Mapping.” IIT Bombay.

I. INTRODUCTION

Petitioner Realtek has failed to prove that any of the challenged claims 1-9 of Patent Owner ATI's U.S. Patent No. 7,742,053 are unpatentable for at least the following reasons.

Ground 1 fails for all challenged claims because Stuttard does not disclose or render obvious the following limitations required by each challenged claim:

- a memory that stores pixel and vertex command threads at the same time, in first and second portions of memory;
- storing complete command threads; and
- an arbiter that selects a command thread from the plurality of vertex or pixel command threads.

In addition, Ground 1 fails for dependent claim 7 because Stuttard does not disclose or render obvious that claim's texture processing engine.

Ground 2 fails for all challenged claims because neither Stuttard nor Williams discloses or renders obvious the following limitations required by each challenged claim:

- a memory that stores pixel and vertex command threads at the same time, in first and second portions of memory ; and
- storing complete command threads

- an arbiter that selects a command thread from the plurality of vertex or pixel command threads.

Ground 2 also fails because there is no motivation to combine Stuttard and Williams, nor reasonable expectation of success of combining these two references that take conflicting approaches to graphics processing. In addition, Ground 2 fails for dependent claim 7 because it again relies on Stuttard, and also fails to disclose the additional restrictions of claims 3 and 8.

Ground 3 fails for all challenged claims because neither Williams nor Whittaker discloses or renders obvious the following limitations required by each challenged claim:

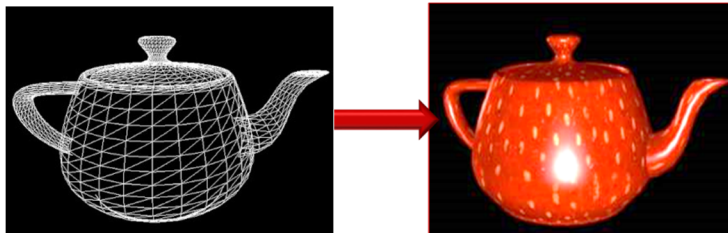
- storing vertex and pixel command threads; and
- an arbiter that selects a command thread from the plurality of vertex or pixel command threads.

In addition, Ground 3 fails because the Petition does not establish a motivation to combine or reasonable expectation of success for the proposed combination, and because that combination fails to disclose or make obvious additional limitations required by claims 2, 3, and 5-8.

II. GRAPHICS PROCESSING

A. Vertex and Pixel Processing

Graphics processing involves manipulating and rendering 3D graphical objects onto a 2D display screen. Ex. 2004 (Real-Time Rendering), 11; Ex. 2009, ¶25. Specifically, 3D objects (typically represented by triangles) are converted into a 2D image (typically comprising pixels) as shown below. Ex. 2003 (Foley), 3, 11-15, 17-20; Ex. 2004, 11-13; Ex. 2009, ¶25. The 2D image represents a depiction of the 3D objects, rendered from the perspective of a 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; Ex. 2009, ¶25.



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. Ex. 2003, 806-809, 866-871; Ex. 2004, 12; Ex. 2009, ¶26. The first stage processes each triangle's corner points, *i.e.*, its “vertices,” and is thus called “vertex processing.” In vertex processing, each vertex's original 3D coordinates are transformed into 2D coordinates for display on a 2D screen. Ex. 2003, 806-809, 866-871; Ex. 2004, 14-21; Ex. 2009, ¶26.

The next stage, “scan conversion,” each 2D triangle is rendered as a collection of 2D dots on the screen, known as “pixels.” Ex. 2003, 806-809; Ex. 2004, 21-22; Ex. 2009, ¶27. The final “pixel processing” stage computes the colors and textures for each rendered pixel. Ex. 2003, 868-872; Ex. 2006 (Catmull); Ex. 2004, 22-23; Ex. 2009, ¶27.

B. Vertex and Pixel Command Threads

As the ITC has found, and as adopted in the Board’s Institution Decision (“ID”), a “command thread” is “a sequence of commands applicable to the corresponding element.” 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; Ex. 2009, ¶46. Thus, a pixel command thread is a sequence of commands applicable to a pixel, and a vertex command threads is a sequence of commands applicable to a vertex.

III. ’053 PATENT

The ’053 Patent discloses novel graphics processing systems and for processing multiple command threads across one or more hardware processing units with the specialized parallel processing architecture of a graphics processor. Ex. 2009, ¶28.

Prior art systems were rigid systems that required providing command threads sequentially. Ex. 1001, 1:31-48; Ex. 2009, ¶29. Prior to the ’053 Patent, systems

consisted of two separate command processing engines: a pixel command processing engine that only ran pixel command threads and a vertex command processing engine that only ran vertex command threads. Ex. 2009, ¶29. Since there were 10x-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. *Id.*

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 command threads and vertex command threads:

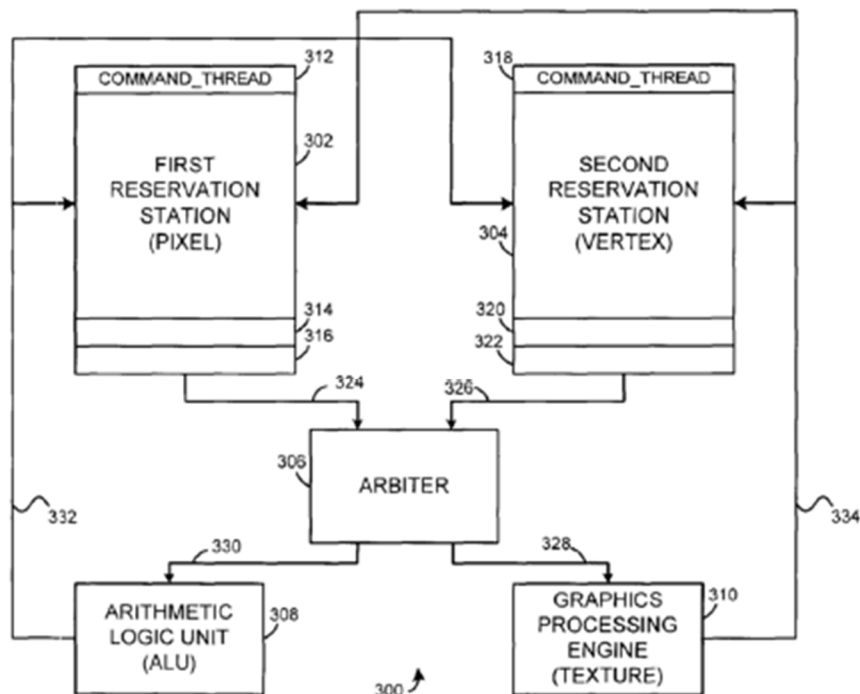


FIG. 4

Ex. 1001, Fig. 4; Ex. 2009, ¶30.

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, 2:45-48, 3:29-36; Ex. 2009, ¶31. Thus, the arbiter ensures 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. Ex. 1001, 3:16-20; Ex. 2009, ¶31.

IV. EXEMPLARY CLAIM

Independent claim 5 demonstrates several of the important distinctions between the inventions of the '053 patent and the prior art:

A graphics processing system comprising:

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*;

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; and

a plurality of *command processing engines*, coupled to the arbiter, each *operable to receive and process the command thread*.

V. CLAIM CONSTRUCTION

A. Storage of “command threads”

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.” Ex. 2009, ¶47. The express language requires that actual “command threads” are stored, not merely a partial sequence of instructions that do not constitute a complete command thread. Accordingly, the claims should be construed to require storing full and complete “command threads,” and not incomplete sequence of instructions from a command thread. Ex. 1001, Claims 1, 5.

Petitioner cannot reasonably contest this because it proposed this same construction in the ITC Investigation, where it was adopted by the ALJ. Ex. 2002 (ITC Markman), 32-39. Petitioner argued that “a complete thread must be stored at a given moment in time” and that it is incorrect that “portions of a complete thread are sufficient so long as the entire thread is stored over a period of time.” *Id.*, 32-33. The ALJ agreed. *Id.*, 39.

This construction is also consistent with the specification, which in all instances describes the storage of a “command thread,” and not merely some incomplete set of instructions thereof. Ex. 2009, ¶48. For example, the specification describes reservation stations as “any type of memory device capable of reserving

and storing *command threads*.”¹ Ex. 1001, 2:39-41. Both Figures 2 and 4 embodiments describe “command threads” as being stored. *Id.*, 2:63-3:36, 3:62-4:64, Figs. 2, 4. Figs. 6 and 7, which illustrate flowcharts of exemplary methods, also provides steps of retrieving “a selected command thread.” *Id.*, 5:43-6:54, Figs. 6-7. Additionally, the specification describes that complete command threads are stored in the memory device even if some instructions from that command thread have already been executed. *Id.*, 4:21-27; Ex. 2009, ¶49. Thus, a command thread is more than a command or an instruction, but rather a sequence of commands/instructions, and the entire command thread must be stored in memory. Ex. 1001, 2:41-45.

Thus, the claims 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”

The explicit language of the claims, the specification, the ALJ’s prior construction, and the testimony of Petitioner’s own expert all establish that this claim

¹ Unless otherwise noted, all emphases are added.

limitation requires the storage of both pixel and vertex command threads at the same time. Ex. 2009, ¶¶51-54.

The plain language of the claims provides that the memory device stores both command thread types at the same time, requiring that it be “operative to store a plurality of pixel command threads *and* ... to store a plurality of vertex command thread.” Ex. 1001, Claims 1, 5.

The '053 Patent specification likewise uniformly describes that the memory device stores both types of command threads at the same time. *Id.*, 2:65-3:3, 3:40-44, 3:50-53, 3:62-4:48, 5:10-16, 5:43-56; Figs. 2, 4, 5; Ex. 2009, ¶52. For example, the specification describes that command threads are stored in one portion of the memory device while vertex command threads are stored in another portion of the memory device. Ex. 1001, 4:5-13. Using priorities, 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 for the arbiter to selectively retrieve either type of command thread from the memory device, both must be stored in the memory at the same time. *See id.*

Indeed, the claims require the same; the arbiter must “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; Ex. 2009, ¶53. Petitioner’s expert testified that “select” means “look[] at *command threads that are available* and

chooses one.” Ex. 2010 (Pedram IPR Depo.), 196:24-197:4. Yet if 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. Ex. 2009, ¶53. The same is true if only vertex command threads were available. *Id.* 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. *Id.* Indeed, Dr. Pedram admitted that an arbiter that is operable to select only from pixel command threads or only vertex command threads does not meet the claims. Ex. 2007, 215:20-24.

In the ITC, the ALJ likewise held that “it is a correct claim construction . . . the memory device(s) are operable to *store both pluralities at the same time.*” Ex. 2002, 84.

Therefore, properly construed, 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”

As discussed, this limitation confirms and further requires that *both* types of command threads must be available to select from at the time when the arbiter’s

selection occurs, *i.e.*, they must be present at the same time in memory. Ex. 2009, ¶55.

Petitioner's Dr. Pedram agreed, testifying that an arbiter that is only operable to select pixel command threads and is not operable to select vertex command threads does not meet the claims. Ex. 2010, 215:20-24.

Petitioner nonetheless contradicts the express claim language, the specification, the ALJ, and its own expert to argue that interpreting this limitation to require a plurality pixel command threads and vertex command threads be stored at the same time so that the arbiter may choose between them somehow renders the phrase "from either of" meaningless. Pet., 12; Ex. 1004, ¶44. It is Petitioner's interpretation which renders the phrase meaningless because "from either of" specifies that the arbiter must be able to select between two possible alternatives, either a pixel type or a vertex type. Ex. 2009, ¶55. If only one type of command thread is present, it cannot select "from either of" the two, it can only select from one type. *Id.*

D. "command processing engine"

Claims 2, 5-9 require a "command processing engine" to receive whichever of the pixel or vertex command threads is selected by the arbiter. Ex. 2009, ¶56. 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. Ex.

2009, ¶57. Accordingly, the command processing engine should be construed as being able to receive and process both pixel and vertex command threads, and not encompassing specialized engines that can only process one type of command thread. *Id.*

Petitioner's expert, Dr. Pedram, agreed that that the unified shader, the subject of the '053 Patent, includes "a command processing engine that processes both pixel and vertex threads." Ex. 2005 (ITC Hearing), 988:20-23. This requirement is explicit in claims 5-9. Ex. 2009, ¶57. And although claim 2 does not explicitly recite that its command processing engine must process in addition to receiving the different types of command threads, a POSITA would understand an arbiter would not provide a command thread to a command processing engine which could not process it. Ex. 2009, ¶58. Petitioner's expert, Dr. Pedram agreed that "each of the claim[ed] processing engines must be operative to process whichever thread is provided to it by the claimed arbiter, be it a pixel command thread or a vertex command thread." Ex. 2005, 988:7-19.

The Federal Circuit has already held the same, recognizing that the "[t]he '053 Patent describes a multi-thread[ed] (unified) graphics processing system that processes both vertex and pixel operations." *ATI Techs. ULC v. Iancu*, 920 F.3d 1362, 1365-6 (Fed Cir. 2019). "The system employs a memory device for storing command threads and an arbiter for providing a command thread to a command

processing engine, . . . whereby the command processing engine performs either vertex or pixel operations based on the command thread from the arbiter.” *Id.*, 1366.

Dr. Pedram agreed that the specification of the ’053 Patent does not discuss or disclose specialized command processing engines, *i.e.*, command processing engines which process only pixel or only vertex command threads. Ex. 2010, 219:24-220:2; Ex. 2009, ¶59. Because “the claims cannot be of broader scope than the invention that is set forth in the specification,” the command processing engines must be able to process both types of command threads. *Gemalto S.A. v. HTC Corp.*, 754 F.3d 1364, 1369 (Fed. Cir. 2014); *see also Retractable Techs., Inc. v. Becton*, 653 F.3d 1296, 1305 (Fed. Cir. 2011) (“In reviewing the intrinsic record to construe the claims, we strive to capture the scope of the actual invention, rather than . . . allow[ing] the claim language to become divorced from what the specification conveys is the invention.”).

Therefore, “command processing engine” 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.

VI. PETITIONER HAS FAILED TO ESTABLISH THE REQUIRED REASONABLE LIKELIHOOD OF SUCCESS FOR ANY GROUND

A. Ground 1: Stuttard In View of the Knowledge Of A POSITA

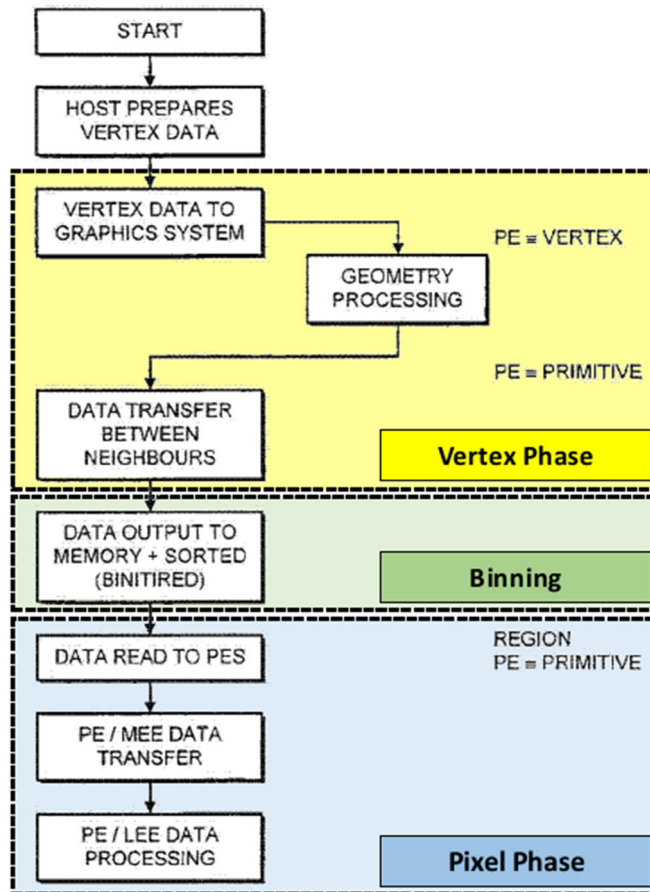
Petitioner alleges that Stuttard in view of the knowledge of a POSITA renders the challenged claims obvious. However, Stuttard does not disclose the claimed “at least one memory device” because the alleged memory device does not store both pixel and vertex command threads at the same time, nor does it store pixel and vertex command threads in first and second portions of memory, respectively, which are requirements of all challenged claims. Ex. 2009, ¶61.

Petitioner admits that Stuttard does not disclose the claimed memory device, but argues that such memory devices were known in the mind of a POSITA and a POSITA would have been motivated to use such a memory device in Stuttard to store command threads, either instead of or in addition to the cache memory used in Stuttard to store commands.

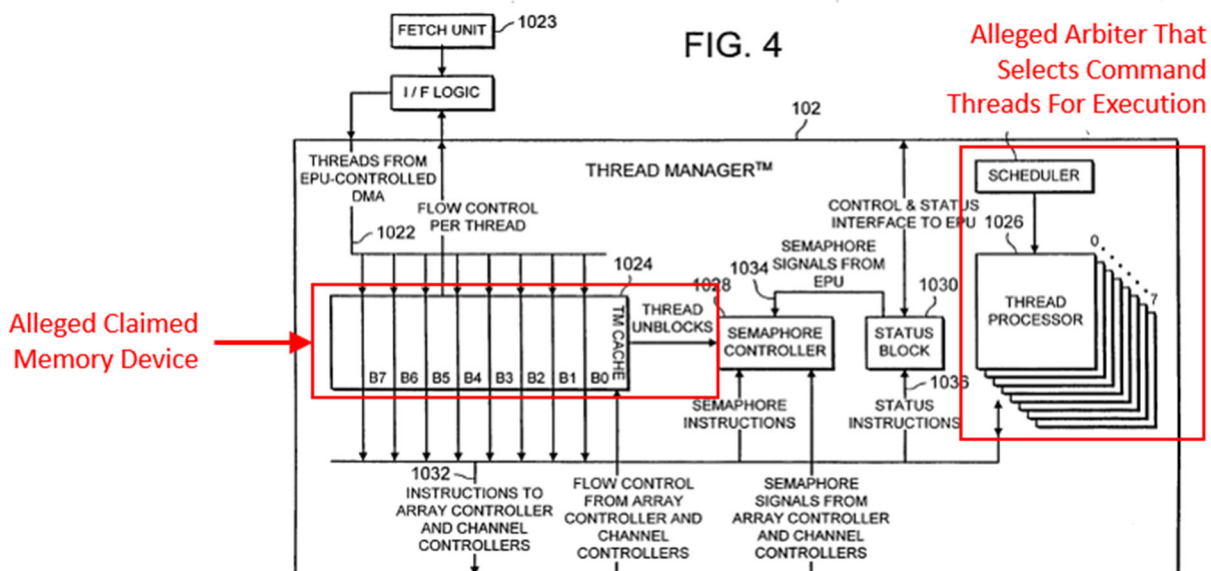
But even setting aside whether Petitioner has proven that the claimed memory device was known, which it has not, there is no motivation to make that change in Stuttard for 3 main reasons.

First, it first operates exclusively on vertices in a first phase. Ex. 2009, ¶¶32-34. Then, only when that vertex phase is fully complete, and after a binning phase, does it operate exclusively on pixels, with no overlap in the execution of the vertex

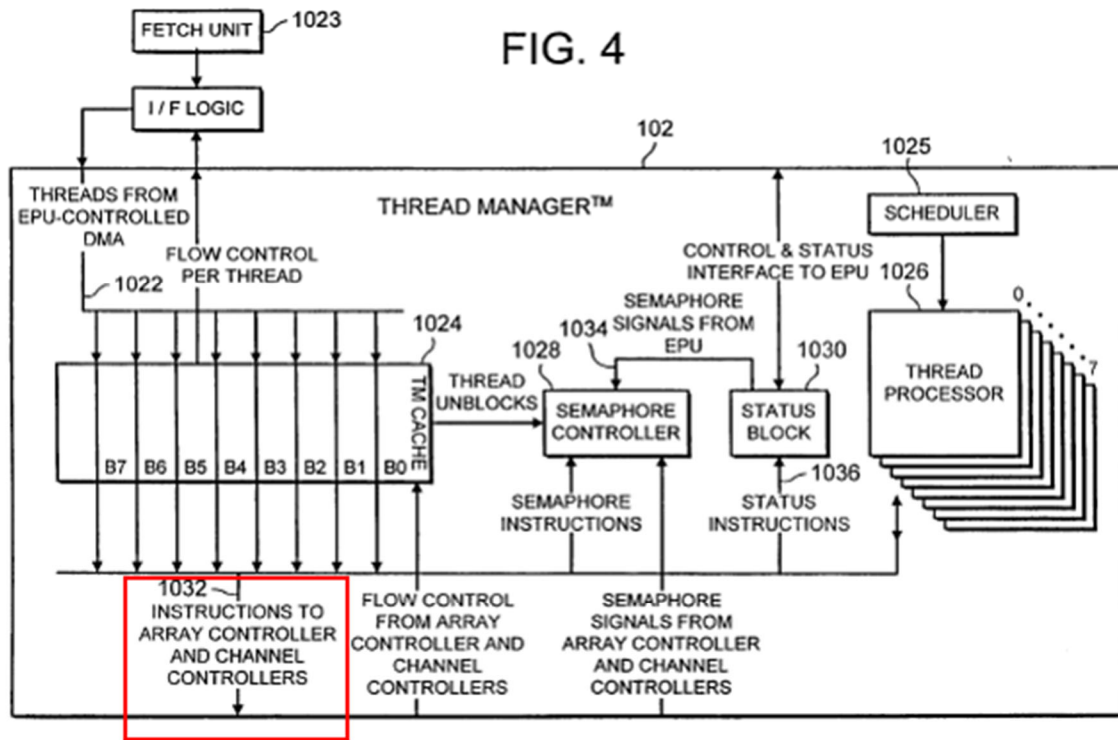
and pixel phases. *Id.* Figure 9 of the reference discloses the flow of graphics processing in Stuttard, and as the annotations demonstrate, the process describes distinct pixel and vertex phases, with an intermediate binning phase separating them:



Ex. 1005, Fig. 9 (annotated). There is thus no reason to modify Stuttard to add a memory, either as a replacement for or addition to cache 1024 of Stuttard, that stores pixel and vertex commands at the same time, as would waste valuable memory resources storing commands that are not being executed in a given phase.



Second, even if the above is incorrect, Stuttard's system can only operate on one instruction at a time. It therefore would not store both type of command threads, as it cannot execute both types of commands simultaneously. Petitioner's theory revolves relies on and requires Figure 4 of Stuttard and the components it describes. As Stuttard itself explains, "In the example shown in Figure 4 the thread manager includes one thread processor 1026 for each thread. . . In *this* particular example the same instruction stream is supplied to all of the processing blocks in the system." Ex. 1005 at 11:27-35.



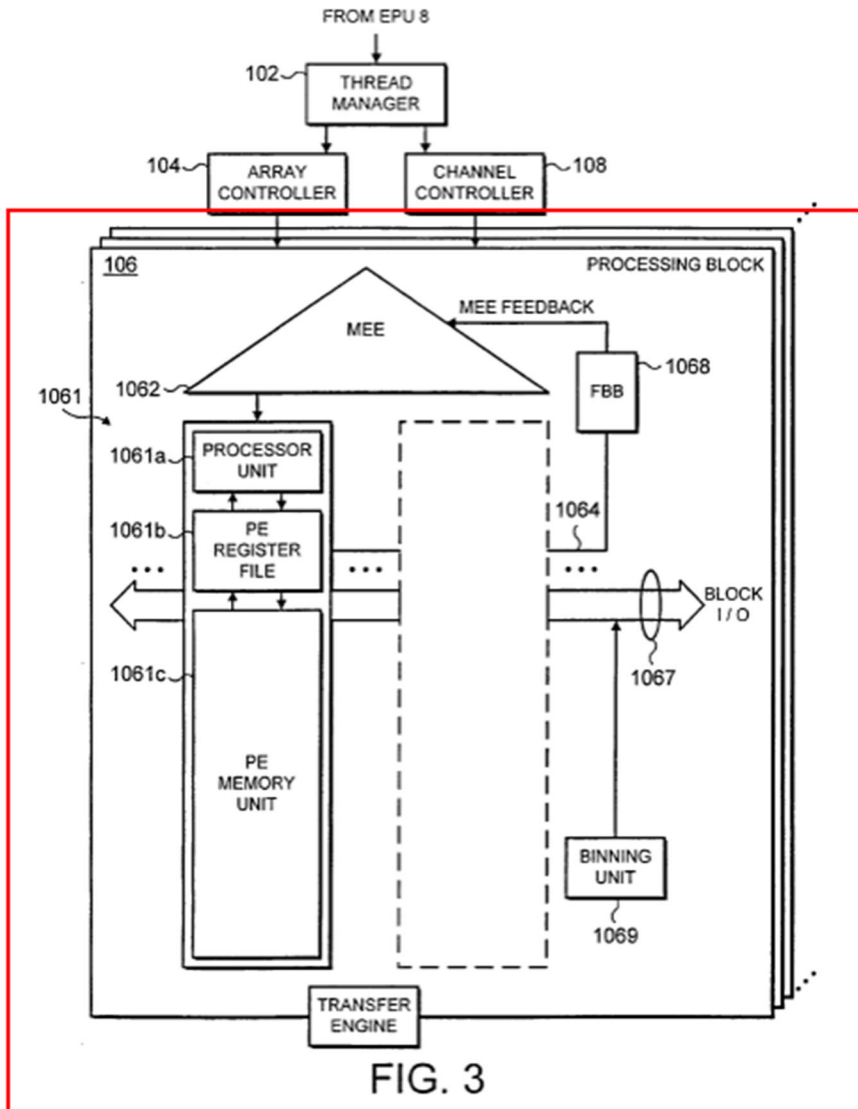
Ex. 1005, Fig. 4 (annotated).

Third, even if the above are both incorrect, the thread scheduler, which “determine[s] which thread should be active at any one time” (Ex. 1005 at 12:12-17) does not know or care about the command thread type. It just cares about priority of the thread and directs the processing blocks to execute the highest priority thread. Petitioner’s expert testified that the thread scheduler’s “selection is not . . . on the basis of vertex versus pixel command threads. . . It just looks at the priority of the thread.” Ex. 2010, 318:11-319:17. There is therefore no reason to replace cache

1024, or add to it, with a memory that simultaneously stores both command thread types in first and second portions. It does not benefit the thread scheduler's selection, or any other aspect of Stuttard, for this more complex, more restrictive, more wasteful memory.

Also, as explained further below, cache 1024 of Stuttard also does not store complete command threads, which is required by all challenged claims.

Further, dependent claim 7's "texture processing engine" is nowhere to be found in Stuttard;



Ex. 1005, Fig. 3 (annotated).

All Petitioner has pointed to in Stuttard is the discussion of some commands that might or might not require texture processing, but there is no indication texture processing is required, much less that a texture processing engine exists in Stuttard.

1. All Claims: Stuttard Does Not Disclose or Render Obvious a Memory Device That Stores a Plurality of Pixel Command Threads and Vertex Command Threads at the Same Time in First and Second Portions

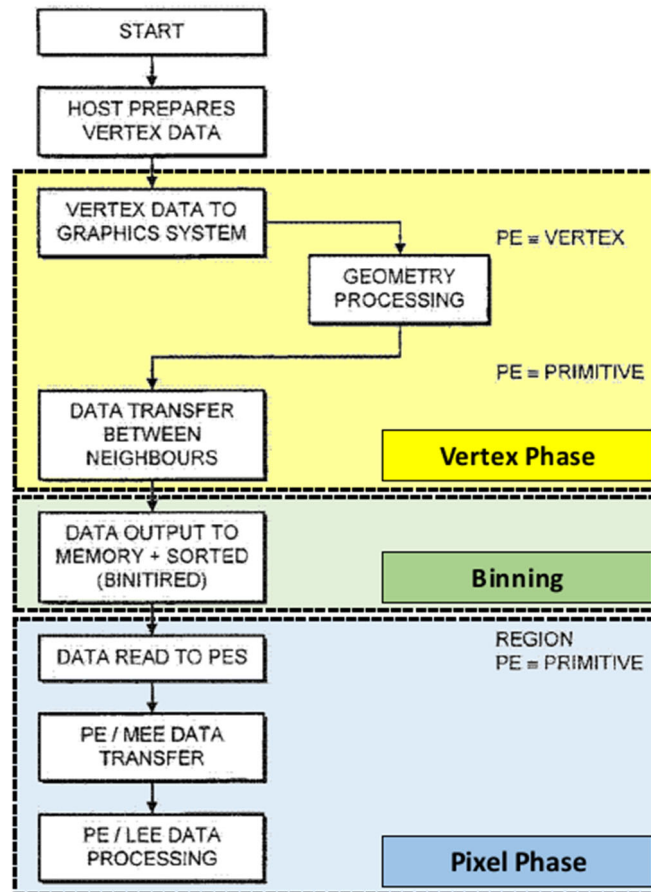
Claims 1 and 5 require “at least one memory device comprising a first portion operative to store a plurality of pixel command threads and a second portion operative to store a plurality of vertex command threads.” Ex. 1001, Claims 1, 5. This claim limitation requires the storage of pixel command threads and vertex command threads in different portions of memory at the same time, and in first and second portions. *See* Section V.B.

Petitioner identifies the cache unit 1024 as disclosing the claimed “at least one memory device” Pet., 20. However, even Petitioner 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.* (emphases in original). Petitioner relies on an alleged “obvious” modification to the memory device to store both types of command threads and to do so in separate “portions.” Pet., 20-25; Ex. 2009, ¶63.

There is no motivation for Petitioner’s proposed modification, as explained above and detailed further below.

a. Stuttard Operates in a Phased Manner

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.”); Ex. 2009, ¶65. Petitioner nor its expert does not even address the binning aspect of Stuttard, which is fatal to its arguments.



Id., Fig. 9 (annotated).

Figure 9 explains that the first step of graphics processing is that the host prepares vertex data. *Id.*, 20:25-30; Ex. 2009, ¶66. Then, vertex data is “loaded into

the PEs of the graphics system so that each PE contains data for one vertex.” Ex. 1005, 20:32-33. Each PE processes 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.* Dr. Pedram testified that 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. Ex. 2010, 231:10-24 (“the vertex processing . . . did not make use of the binitization process and the region-based processing”).

The binning process mandates the processing of vertices and pixels in distinct ***block parallel*** phases. Ex. 2009, ¶67. First, Stuttard explains that binning “must maintain primitive order between the geometry and rasterisation phases due to requirements of most host systems.” Ex. 1005, 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.* Stuttard also 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.

While each processing block has a binning unit, all binning units work collectively to complete the binning process. Ex. 2009, ¶68; Ex. 1005, 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.

After binning, other processing must occur before the processing blocks can begin to process pixel instructions. Ex. 2009, ¶69. First, the primitive information is read back into individual PEs, such that “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. *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, 234:22-235:7. Only after this scan conversion can the pixel data be processed. Ex. 1005, 22:26-32.

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. Ex. 2009, ¶70. Dr. Pedram agreed that before binning, there's no region-based processing, meaning that vertex processing is not region-based. Ex. 2010, 230:24-231:9. He further agreed that Stuttard's pixel processing is region-based and "cannot occur until the vertex processing for all the vertices has been done because that is the stage that will determine where the primitives will be." *Id.*, 239:17-240:1. 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. Ex. 2009, ¶70; Ex. 1005, 21:18-21. Even after binning, the primitives must undergo scan conversion before any pixel processing can begin. Ex. 2009, ¶70. Thus, Stuttard's phased system stands for more than the proposition that one must have a pixel to shade before shading a pixel. *Id.*

Dr. Pedram's testimony that Stuttard may start processing pixels before vertices are completed is wrong, and divorced from any disclosures in Stuttard. *E.g.*,

Ex. 2010, 236:11-237:24; Ex. 2009, ¶71. While it is “possible,” Ex. 2010, 237:18, that some graphics processing systems work 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. Ex. 2009, ¶71. 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. *Id.*

There is also no circumstance in which a POSITA would understand that Stuttard’s system diverts processing power to vertices for the next frame prior to completion of pixel processing for the prior frame. Ex. 2009, ¶72. This is especially true given that processing of pixels takes more resources than the processing of vertices. *Id.* 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 present phase as soon as possible. *Id.* Nor would it benefit Stuttard’s alleged arbiter because, according to Dr. Pedram, the alleged arbiter is blind to thread type during the selection process. Ex. 2010, 318:11-320:7.

Making such modifications would be antithetical to rendering frames in the shortest amount of time possible. Ex. 2009, ¶73. 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. *Id.* 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. *Id.*

b. Stuttard Executes One Instruction at a Time

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. 1005, 2:17-23. A SIMD is only capable of processing a single thread at any given moment in time. Ex. 2009, ¶74.

Stuttard makes disclosures that its single SIMD array system is capable only of processing the same instruction across the entire array of processing blocks. Ex. 2009, ¶75. 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.

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; Ex. 2009, ¶76. 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." Ex. 1005, 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. Ex. 2009, ¶76.

The entire array of processing blocks within Stuttard's system is controlled by the thread manager. Ex. 2009, ¶77. The main purpose of the thread manager is to control which thread is currently active, and how to switch from one thread to another. *Id.* 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.

Petitioner 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; Ex. 2009, ¶78. This is directed at how the thread processors within the thread manager operate, and not the processing blocks. *Id.* 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 currently executing active thread stalls. Ex. 2009, ¶78. 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 the processing blocks in the system.” Ex. 1001, 11:33-35.

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. Ex. 2009, ¶79. These benefits explain the division of the SIMD array of processing elements into processing blocks. *Id.* 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. Ex. 2009, ¶79.

Accordingly, Stuttard provides the processing blocks can operate on different regions of the display, *i.e.*, different data groups. Ex. 2009, ¶80. 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. 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. Ex. 2009, ¶80.

Dividing the array of processing elements into processing blocks also allows for redundancy. Ex. 2009, ¶81. 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. Ex. 2009, ¶81.

Further, the ID misreads claims 103-105 of Stuttard, which it cites as evidence of invalidity, but Petitioner does not rely on. Ex. 2009, ¶82. 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. *Id.* 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.

Second, Claim 103’s SIMD array of PEs is further divided into processing blocks that process respective groups of *data* items. Ex. 2009, ¶83. 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. *Id.* The SIMD design means that the same instruction are being

performed on the different groups of data items. *Id.* Meanwhile, the local memory unit is for storing ***data items***. Stuttard distinguishes data from instructions/threads by describing data as “graphics data” or “video data.” Ex. 1001, 4:5-9. Similarly, the segment register stores address information for those data items; it does not store instructions or threads. *Id.*

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. Ex. 2009, ¶84. 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. *Id.* 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

Petitioner’s argument is Stuttard’s system would benefit from knowing where in memory commands of specific types are, because it would create “faster and convenient storing and retrieving of different types of data,” “faster access to different types of command threads.” Pet., 20, 22; Ex. 2009, ¶85. But as Petitioner’s expert, Dr. Pedram, has now conceded, the scheduling module 1025 of Stuttard, which along with the thread processors 1026, is the claimed arbiter that performs the claimed “select[ing]” of the command thread to be processed, does not perform that

selection on the basis of command thread *type*: “its selection is not on the basis of -
- as far as I remember -- the selection it makes, it’s not on the basis of vertex versus
pixel command threads. . . It just looks at the priority of the thread, looks at the ready
versus yielding status.” Ex. 2010, 318:11-319:17. Why, then, would a POSITA go
through the trouble of modifying the memory of Stuttard, which works for its
purposes, to place particular command types in particular first and second memory
locations, creating overhead and inefficiencies in memory storage and access, indeed
slowing down the process by which the arbiter, which is agnostic to command type,
access the data in memory? He or she would not. Ex. 2009, ¶85.

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

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; Ex. 2009, ¶86. 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.* Courts construe
“operative to” or “operable to” to be synonymous with “configured to.” *E.g., TQ
Delta, LLC v. Commscope Holding Co.*, No. 2:21-CV-310-JRG, 2022 U.S. Dist.
LEXIS 102577, at *28 (E.D. Tex. June 8, 2022) (“[T]he terms ‘operable’ and
‘operable to’ should be construed to refer to being configured to operate in the

recited manner.”). This means meaning that the system is configured to actually perform a function, not merely that it is capable of doing so. *Aspex Eyewear, Inc. v. Marchon Eyewear, Inc.*, 672 F.3d 1335, 1349 (Fed. Cir. 2012). That is, mere probability or happenstance is insufficient. Ex. 2009, ¶87. Even if the Board disagrees with the above arguments from Patent Owner, it remains the case that there has been no showing of 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

Claims 1 and 5 of the '053 Patent require a memory device storing “command threads,” and an arbiter selecting “command threads.” Petitioner has failed to show that Stuttard’s cache memory unit stores complete threads. As explained above in Section V.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. Stuttard only discloses the storage of instructions for a thread, which does not indicate that all the instructions for each thread are stored at any one time. In addition, Dr. Pedram provided testimony regarding the capabilities of the alleged arbiter undermining his opinion; he first testified the claimed arbiter must know what type of command thread it is selecting, yet opined that he didn’t know whether Stuttard’s arbiter did so.

Petitioner points to cache unit 1024 in Stuttard as the claimed “memory device.” Pet., 20. But cache unit stores instructions, *i.e.*, commands. *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; Ex. 2009, ¶89. Further, Stuttard discloses that instructions, not complete threads, are fetched by thread manager from the host system and sent to cache unit for storage. *Id.* “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.” Ex. 1005, 8:17-24. All these disclosures provide that instructions are fetched and stored in cache unit, and not complete threads. Ex. 2009, ¶89. There is no disclosure that all the instructions for a thread are fetched and stored by cache unit at the same time. *Id.*

Dr. Pedram’s earlier testimony indicates that Stuttard does not explicitly disclose storing complete threads:

Q So at any given moment in time --

A Yes, sir.

Q -- the cache unit 1024 might never store all of the instructions in a thread?

A May or may not.

Ex. 2008 (Pedram ITC Depo.), 183:6-10. As Dr. Pedram provides, this is “what happens in these kinds of systems. It’s called pre-fetch and execute, and you try to overlap, therefore, the actual fetch and execute operation.” That is, the process of retrieving the commands overlaps with the execution of other commands of the same thread, which is why Dr. Pedram answered that cache unit 1024 may never store all commands in a command thread. The ID stated that Patent Owner failed to note the context for Dr. Pedram’s testimony and that it was about “these kinds of systems.” ID, 30. To the extent the ID was indicating the possibility that the testimony was not about Stuttard, that is incorrect, as the question posed to Dr. Pedram was specifically about Stuttard. Ex. 2008, 181:15-21.

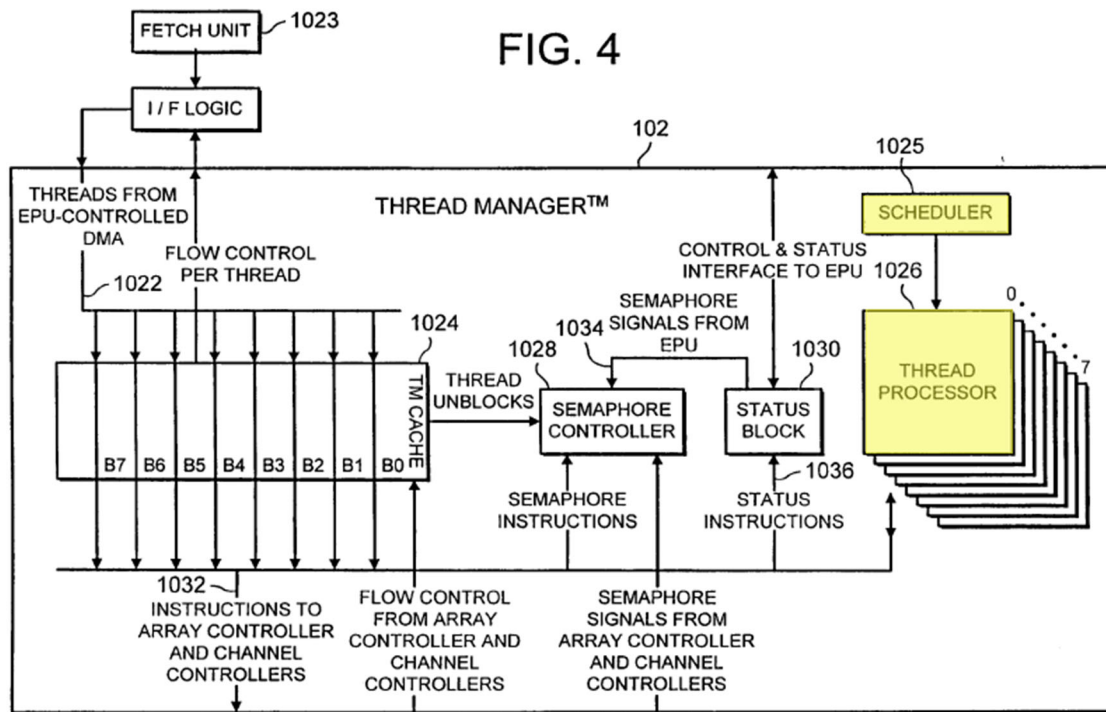
The ID found that Patent Owner’s evidence “do[es] not suggest that partial threads are fetched.” ID, 29-30. The evidence described above states that instructions, which, like commands, are partial threads (Ex. 1001, 2:41-45), are what are fetched and stored in cache memory; Ex. 2009, ¶89. The evidence does not state that threads are fetched and stored in memory and Dr. Pedram’s testimony further establishes that complete threads are not necessarily stored. It is Petitioner’s burden to establish the reference discloses the claim limitations, and it has not done so. *In re Magnum Oil Tools Int’l, Ltd.*, 829 F.3d 1364, 1377 (Fed. Cir. 2016) (burden improperly on the patentee to prove non-obviousness).

Additionally, the addition of the APA or any knowledge of a POSITA does not resolve this deficiency, nor has Petitioner argued so. Petitioner argues only that the APA teaches storing different types of command threads separately. Pet., 22-25.

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

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; Ex. 2009, ¶91. At a minimum, the claim language itself requires the arbiter select from a pool of two pixel command threads *and* two vertex command threads.

Petitioner argues that “thread scheduler 1025 and thread processors 1026 together correspond to the claimed ‘arbiter’ of claims 1, 2, and 5.” Pet., 25. Petitioner relies solely on the fact that “[t]he Board has already concluded that the thread manager 102, which includes a thread scheduler 1025 and thread processors 1026 satisfy the arbiter limitations,” in the prior IPR by LG, to conclude that the “arbiter” claim limitation is rendered obvious by Ground 1. *Id.*, 26.



Ex. 1005, Fig. 4 (annotated).

Petitioner’s argument 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; Ex. 2009, ¶93. The alleged arbiter cannot because, as explained above in Section VI.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. Ex. 2009, ¶93. Stuttard’s cache memory stores only vertex instructions during vertex processing, and pixel instructions during pixel processing. *Id.* This is true even if

Patent Owner's claim construction regarding the memory device being required to store both pixel and vertex command threads at the same time is incorrect. *Id.* 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.*

Further, the 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. Ex. 2009, ¶94. 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. Ex. 2009, ¶94.

Further, 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; Ex. 2009, ¶95. 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). *Id.* Even if Petitioner 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. *Id.* 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; Ex. 2009, ¶95. Nor does the APA disclose that the arbiter is coupled to a command processing engine. Ex. 1001, 1:58-64.

Petitioner has not explained how Stuttard’s architecture would be modified to fit the architecture of the APA, or vice versa. Ex. 2009, ¶96.

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

Petitioner argues that texture processing engines are inherent due to Stuttard’s disclosure of performing “lighting and shading.” Pet., 27-28. However, a POSITA would have not have recognized that lighting and shading operations necessarily require a texture processing engine, nor does Petitioner provide any support for this assertion. Ex. 2009, ¶97. Further, Dr. Pedram provided an example where lighting and shading would only require vertex processing, and thus not require a texture processing engine. Ex. 2010, 265:3-8. Inherency may not be established by probabilities or possibilities; the mere fact that lighting and shading may involve

texture processing engines is insufficient. *Persion Pharms. LLC v. Alvogen Malta Operations LTD.*, 945 F.3d 1184, 1191 (Fed. Cir. 2019).

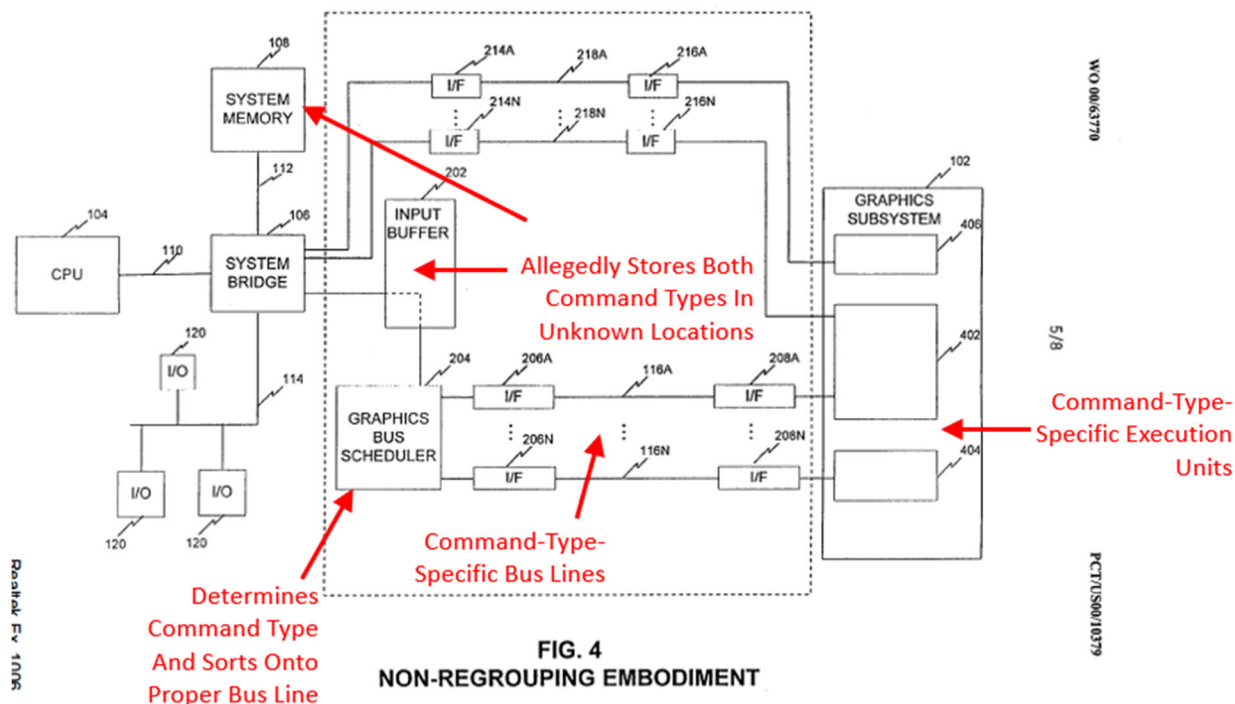
Further, a texture processing engine is more complex than simply processing lighting and shading. Ex. 2009, ¶98. Texture processing engines apply textures to certain portions of an image. *Id.* Processing lighting and shading does not require applying texture, or even if it does, a specialized texture processing engine to apply the texture. *Id.* 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), 114 (discussing disabling a texture and code for rendering an object without texture), 95 (Figure 4.2), 107 (Figure 5.7), 94-97, 107-109 (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. Ex. 2009, ¶98.

B. Ground 2: Stuttard In View Of Williams and The Knowledge Of A POSITA

In Ground 2, Petitioner proposes to use Williams to solve Stuttard's lack of disclosure of the claimed memory device. Pet., 40. But Williams fails to disclose

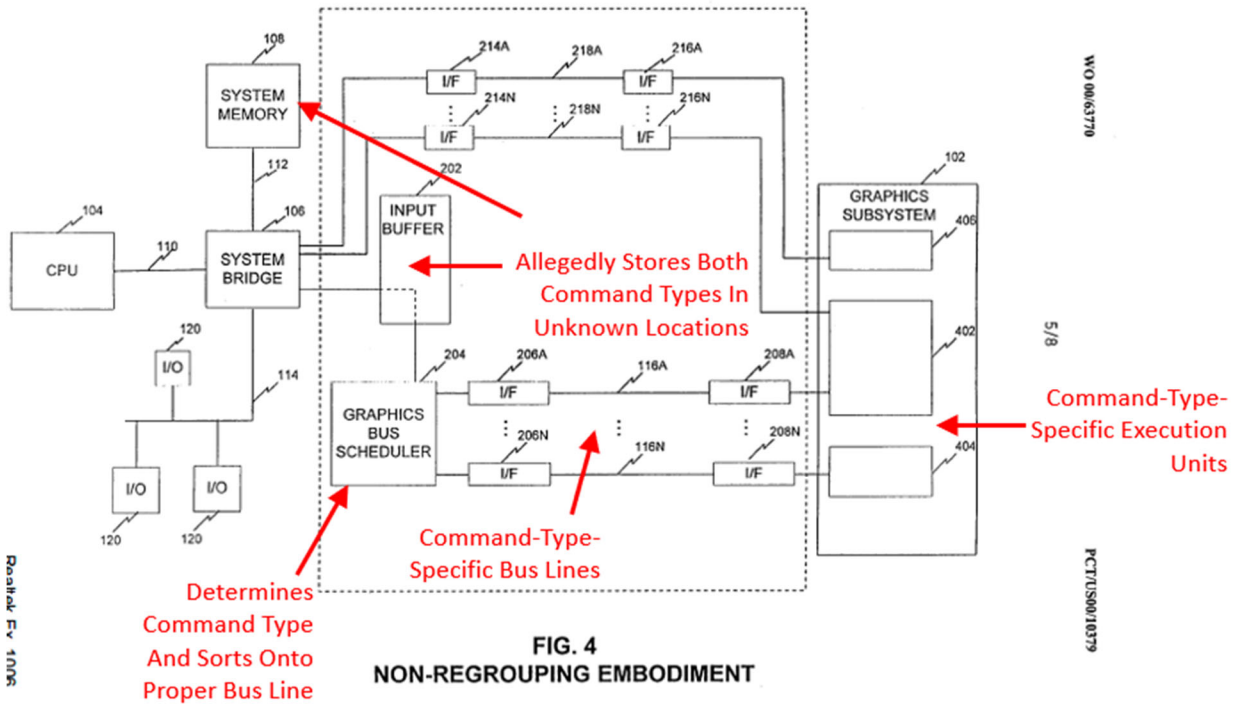
the memory device because it too fails to disclose storing command threads, and it fails to disclose storing vertex and pixel command threads in first and second portions of memory, as Petitioner admits, instead relying on a modified version of the Williams memory device. There is no motivation to make the modification Petitioner proposes to the memory in Williams, and even if there were, there would be no reason to include such a modified memory in Stuttard. Further, Williams and Stuttard take completely different approaches to graphics processing and are incompatible with one another.

First, Petitioner admits that no memory in Stuttard stores both types of command threads in first and second portions. Petitioner proposes modifying one or more of those memories, primarily input buffer 202, so that they do, arguing that it would make the system more efficient to know where particular command types are located. But it would not; such restrictions on memory storage location would make Williams slower, more wasteful, less flexible, and less efficient and it would provide no benefit because there *is* no benefit to knowing where particular command types are stored in memory in Williams. The only relevance in Williams of what *type* a certain command is, is which bus to put it on. That function is already performed by the Graphics Bus Scheduler. There is thus and thus no reason to modify Williams's memories with a more complex, less efficient, slower one:

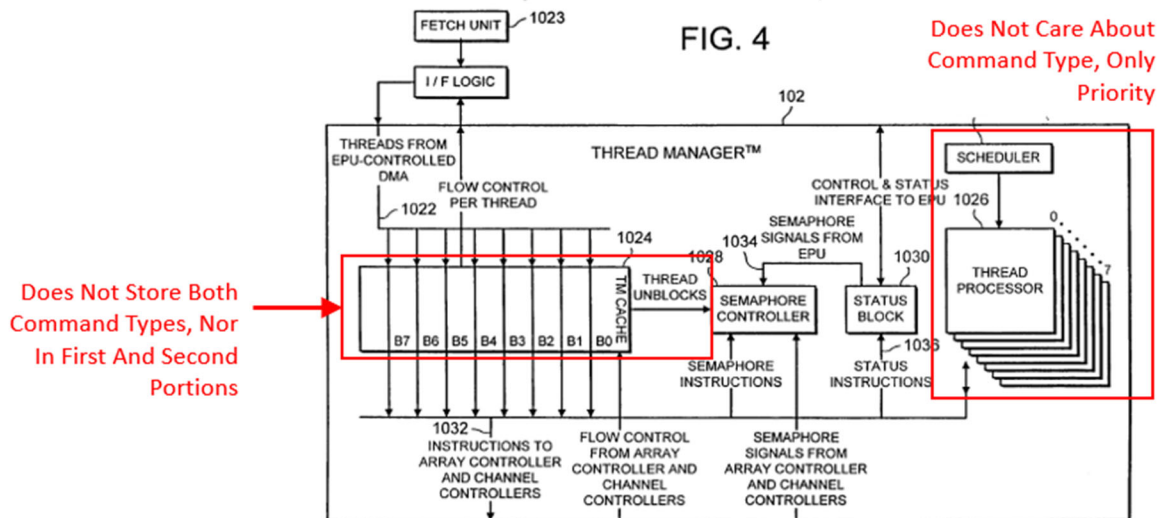


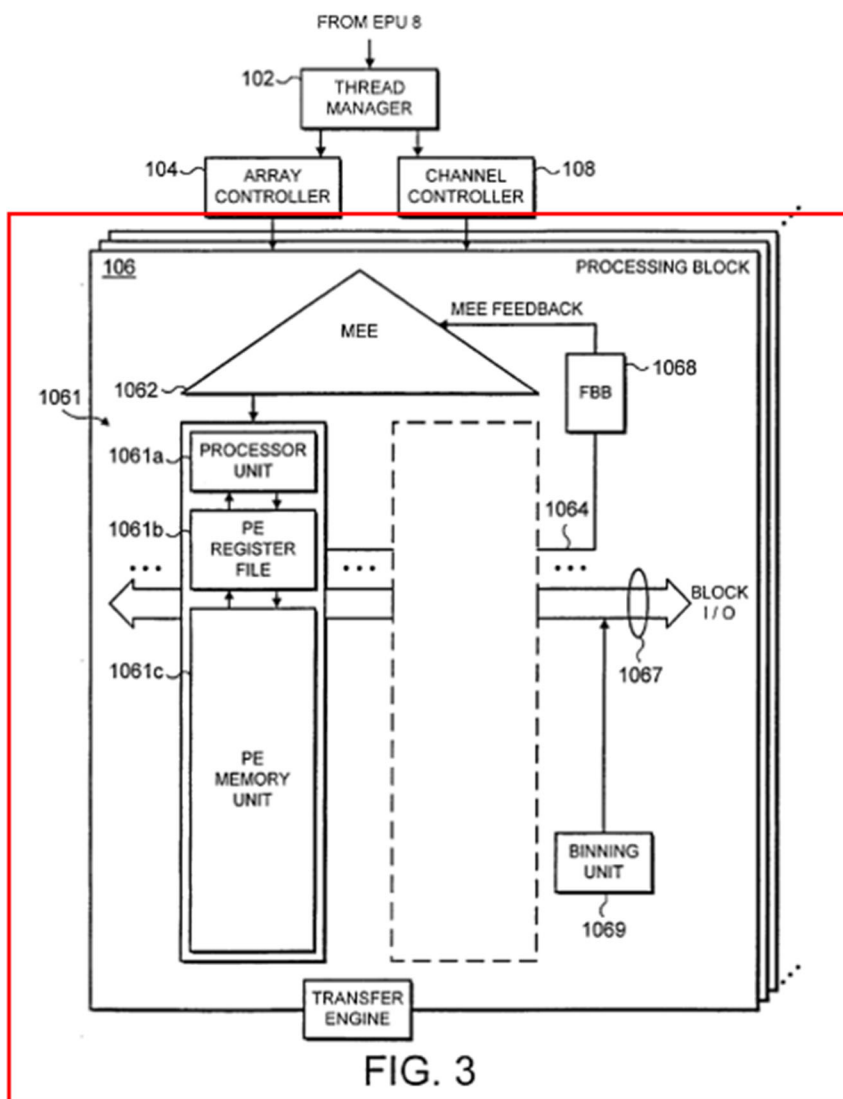
Second, even if the Board disagrees with the above, similarly to Ground 1, there is no reason to include any of Williams's components in Stuttard—not the memory, not the command-type-specific bus lines, not the command-type-specific execution units—into Stuttard, a system that according to Petitioner's expert selects commands for execution based solely on priority and not in any way on the basis of command type, and sends them to execution units that can allegedly execute any type of command at any time.

Williams:



Stuttard:





Benefits From
Alleged Capability To
Any Type Of
Command At Any
Time Per Petitioner

Ex. 1005, Figs. 2-3 (annotated). Petitioner has presented no true motivation to combine other than baseless arguments regarding speed and efficiency. But given Stuttard's operation, it would be slower and less efficient to add any of the components of Williams to Stuttard. It would waste memory space, perform unnecessary command type categorization, and/or using rigid bus line designations, which Petitioner's own expert called a bad idea. Ex. 2010, 281:23-283:5.

Additionally, as explained below, Williams does not disclose storing complete command threads at all, in any of the memories Petitioner points to; it only stores individual commands.

1. All Claims: Williams Does Not Disclose or Render Obvious Storing Vertex and Pixel Command Threads in First And Second Portions of Memory

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. 1006, 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; Ex. 2009, ¶111. Petitioner 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.” Pet., 42; Ex. 2010, 285:18-23.

Petitioner’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

Petitioner argues that “it was well known that commands stored in consecutive addresses may be *read and written with fewer instructions* than reading commands

from (or writing them to) non-consecutive addresses because the latter involves address jumps.” Pet., 42. Those hypothetical benefits have no relevance to Williams. Ex. 2009, ¶112. 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 difference between the embodiments occurs after command transmission across the bus lines; the earlier steps regarding how the GBS receives commands is the same in the embodiments. Ex 2009, ¶112.

The Graphics Bus Scheduler 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 from input buffer by GBS in their original order. Ex. 2009, ¶113. Thus, GBS does not read memory addresses to obtain the commands. *Id.* 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. *Id.*

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. Ex. 2009, ¶114. Petitioner apparently argues that somehow it would require *fewer* instructions if the commands were required to be stored in particular locations. *Id.* That is nonsensical: such a change strictly adds overhead and complexity, rather than reducing it. *Id.*

Moreover, there is no benefit to the GBS to store particular command types in particular locations. Ex. 2009, ¶115. The GBS does not care where the commands are. *Id.* 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; Ex. 2010, 287:18-288:18. Then it puts the command on the bus line for that command type. *Id.*, 270:20-271:4, 280:15-281:22. There is no reason to add a pre-sorting requirement such that commands of particular types are stored in first and second memory portions. Ex. 2009, ¶115.

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 Petitioner suggests. Pet., 42; Ex. 2009, ¶116.

In fact, modifying Williams to require reading particular memory addresses in order to retrieve commands would substantially slow the system down. Ex. 2009, ¶117. 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. *id.* 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. *Id.* Having two components with essentially the same function would lessen the efficiency of Williams. *Id.* Therefore, Petitioner 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. *Id.*

Petitioner’s alternative argument, regarding Figure 6 of Williams, fails for the same reasons. Pet., 43; Ex. 2009, ¶117. Petitioner identifies Williams’ input buffer and output buffer of Fig. 6 (reproduced below) as disclosing the “at least one memory device” claim limitation. Pet., 43. Petitioner 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. Ex. 2009, ¶117. Simply put, the claimed arbiter must select *from* the pool of command threads stored by the memory device(s). *Id.* But Petitioner 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. *Id.* 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. Ex. 1006, 23:16-21. Petitioner has provided no indication the same would not be true with respect to both Grounds 2 and 3. Ex. 2009, ¶117.

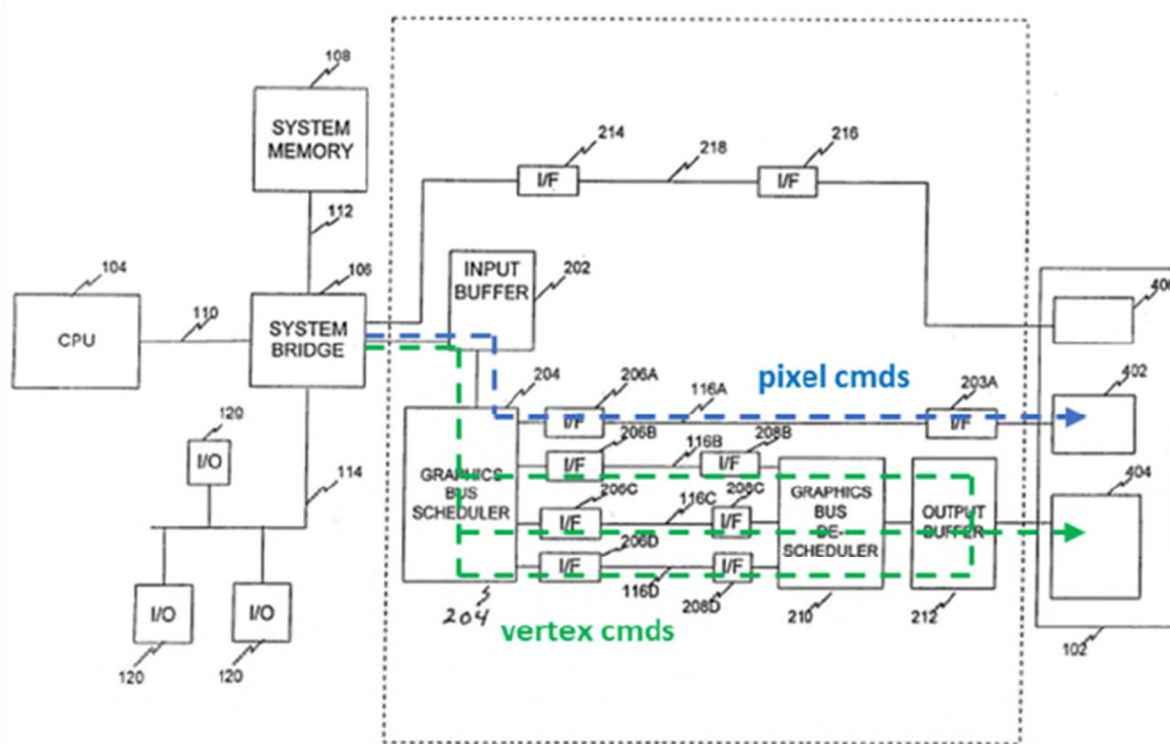


FIG. 6

Id., Fig. 6 (annotated by Petitioner); Pet., 43. Additionally, this argument suffers from all the same arguments above, *e.g.*, this disclosure merely stores commands and not command threads. Ex. 2009, ¶117. 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

For the reasons discussed above in Section VI.B.1.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 first and second portions of memory. Ex. 2009, ¶118. 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, as Dr. Pedram testified. Ex. 2010, 318:11-320:7.

While Dr. Pedram clarifies that he ultimately is only relying on the disclosure of the memory itself in Williams, not the GBS or the bus lines (Ex. 2010, 282:1-7), the ID also considered a theory in which the GBS and the type-specific bus lines of Williams were included in the combination. Ex. 2009, ¶119. But that is not Petitioner's theory, and for good reason, as it would make no sense to do in Stuttard. *Id.* According to Petitioner, Stuttard takes a flexible approach, not concerned with command type, and instead only concerned with command priority, in which any processing element can process anything. *See* Pet., 17, 25; Ex. 2010, 318:11-319:17. 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. Ex. 2010, 270:20-271:4, 280:15-281:22, 287:18-288:18. There is no benefit to adding a GBS to identify command type, and dedicated-command-type bus lines of Williams, into Stuttard. Ex. 2009, ¶119. Stuttard is concerned with maintaining flexibility to accommodate priority differences, and does not care or know about command type in the process of determining which thread to process, as Dr. Pedram admitted. Ex. 2010, 318:11-319:17. Dr. Pedram crystalized this point: when asked, “Why would it be beneficial to have type-specific bus lines in Stuttard?” he answered that he did not propose that modification, and that “because the nature of the applications in Stuttard is more general . . . a fixed . . . assignment of buses between the cache memory and the parallel processing blocks may not be a good idea.” *Id.*, 281:23-283:5. Patent Owner agrees that it would be a bad idea and one that a POSITA would not perform.

2. All Claims: Williams Does Not Disclose or Render Obvious Storing “Command Threads”

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 first and second portions of memory, because Williams does not even disclose commands or command threads claimed by the '053 Patent. Ex. 2009, ¶100. Williams' graphics commands are not pixel or vertex commands, but higher level API commands, such as OpenGL commands. *Id.* 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. *Id.*

Nor has Petitioner 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 VI.A.1.d.

a. Williams' Graphics Commands Are Not Pixel and Vertex Command Threads

Williams does not store "commands" or "command threads." Ex. 2009, ¶102. Williams' graphics commands 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)." *Id.*, ¶103; 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. Ex. 2009, ¶103. 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	glTextImage2D (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) ;

Id., Table 2 (excerpt). Therefore, Williams’ graphics commands differ from the command threads of the ’053 Patent. Ex. 2009, ¶104. 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. *Id.* Depending on how the system was configured, these OpenGL commands may or may not result in vertex command threads and pixel command threads. *Id.* There is not enough information in Williams to say that they would be, nor has Petitioner provided any analysis on this. *Id.* It is dependent on unstated implementation details on how those commands would be carried out. *Id.*

b. Williams Does Not Store Complete Command Threads

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." Ex. 2009, ¶105. 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. Pet., 14. That is, there must be some relationship between multiple instructions such that they are sequenced together, and they must be applicable to particular vertex data or pixel data, to constitute a thread. Ex. 2009, ¶105. 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 V.A.

But nothing in Williams describes storing such a sequence, *i.e.*, a command thread as a whole. Ex. 2009, ¶106. Williams only stores individual graphics commands, and does not even mention the word "thread" or any equivalent concept. *See generally* Ex. 1006. Petitioner 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. Pet., 42; Ex. 2009, ¶106.

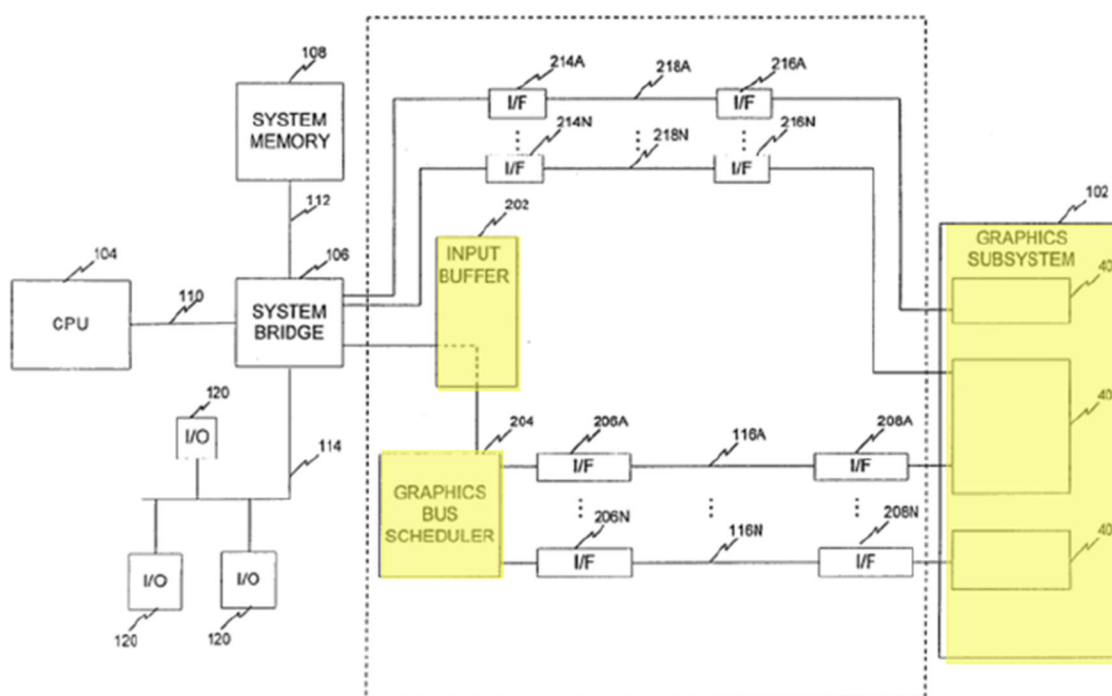


FIG. 4
NON-REGROUPING EMBODIMENT

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. 2009, ¶106; Ex. 1006, 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.*

Further, the storage devices in the Figure 6 embodiment Petitioner relies on in the alternative, also stores commands and not command threads. Ex. 1006, 23:1-

21; Ex. 2009, ¶107. This embodiment also describes transferring “commands” one by one across certain busses. *Id.*

The ID 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. Ex. 2009, ¶108. First, as explained above, 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. *Id.* Second, display list 704 that the ID pointed to is not alleged to be the claimed memory device by Petitioner, nor does Petitioner ever discuss display list 704. *Id.* The ID thus added new arguments not in the Petition. “The ‘Board must base its decision on arguments that were advanced by a party, and to which the opposing party was given a chance to respond.’” *Oren Techs., LLC v. Proppant Express Invs. LLC*, No. 2019-1778, 2021 U.S. App. LEXIS 21859, at *13 (Fed. Cir. July 23, 2021); *Magnum Oil*, 829 F.3d at 1381 (“We have held that the Board erred when it relied on a prior art reference that was unasserted for meeting a particular limitation, even though the passage from the reference that the Board relied on for disclosing the limitation had been block quoted in the petition and institution decision, in a different context.”). There is no evidence that display list

704 meets the other requirements of the claims, *e.g.*, being a memory device, being coupled to the arbiter, etc.

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 (Petitioner's alleged memory device) and GBS (Petitioner's alleged arbiter). Ex. 1006, 23:23-34, 24:5-9, Fig. 7; Ex. 2009, ¶109. 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. *Id.*

Petitioner also fails prove the requirement that the memory device must store at least two pixel command threads and vertex command threads. Ex. 2009, ¶110. As described above in Section V.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. Petitioner does not even address this requirement with disclosures from Williams. *Id.* Therefore, Williams cannot resolve this deficiency of Stuttard.

3. All Claims: Stuttard Does Not Disclose or Render Obvious the Claimed Arbiter

Petitioner relies on the alleged arbiter from Stuttard for this ground. For the same reasons explained above, this ground does not disclose the claimed arbiter. *See* Section V.B; Ex. 2009, ¶120.

Moreover, Petitioner fails to disclose how the same, unmodified arbiter in Stuttard would work when the memory changes as proposed. Ex. 2009, ¶121. For that reason alone, the combination fails.

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

Petitioner identifies an excerpt from Williams in its claim chart with no explanation, stating “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.” Pet., 49 (citing Ex. 1006, 23:11-14). To the extent this identifies input buffer as the “first memory device” and output buffer as the “second memory device,” this theory fails because the output buffer cannot be one of the claimed memory devices. Ex. 2009, ¶¶123-124. As explained above in Section VI.B.1, the alleged arbiter (GBS) does not *select from* the graphics commands stored in the output buffer. Rather, output buffer is where the graphics commands are stored after they have been sorted/sent by GBS. *Id.*

5. Claim 7: Stuttard Does Not Disclose or Render Obvious a Texture Processing Engine

Ground 2 does not disclose or render obvious these claims for the same reasons explain in Section VI.A.4. Ex. 2009, ¶125.

6. No Motivation to Combine or Reasonable Expectation of Success

There is no motivation to combine these references, and they are fundamentally incompatible as explained above, for example in Section VI.B.1.

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. Ex. 2009, ¶128; *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.*

Petitioner also 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. 2009, ¶130. 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. 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 from increasing the speed of transferring items to the graphics processor. Ex. 2009, ¶130. 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. *Id.* There is no indication that bandwidth from the host system to the graphics systems is a limiting factor in Stuttard. *Id.*

Petitioner identifies the issues with AGP buses discussed in Williams and used by Stuttard's system as a reason for a motivation to combine. Ex. 2009, ¶131. Stuttard's system is not limited by bandwidth of the AGP buses. *Id.* A POSITA would not be motivated to increase bandwidth unless it was needed. *Id.* Further, Williams acknowledges that AGP buses have "relatively high bandwidth." Ex. 1006, 7:19-23. Stuttard describes AGP buses as having "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 buses as causing a bottleneck to its system. *Id.* Implementing Williams' invention

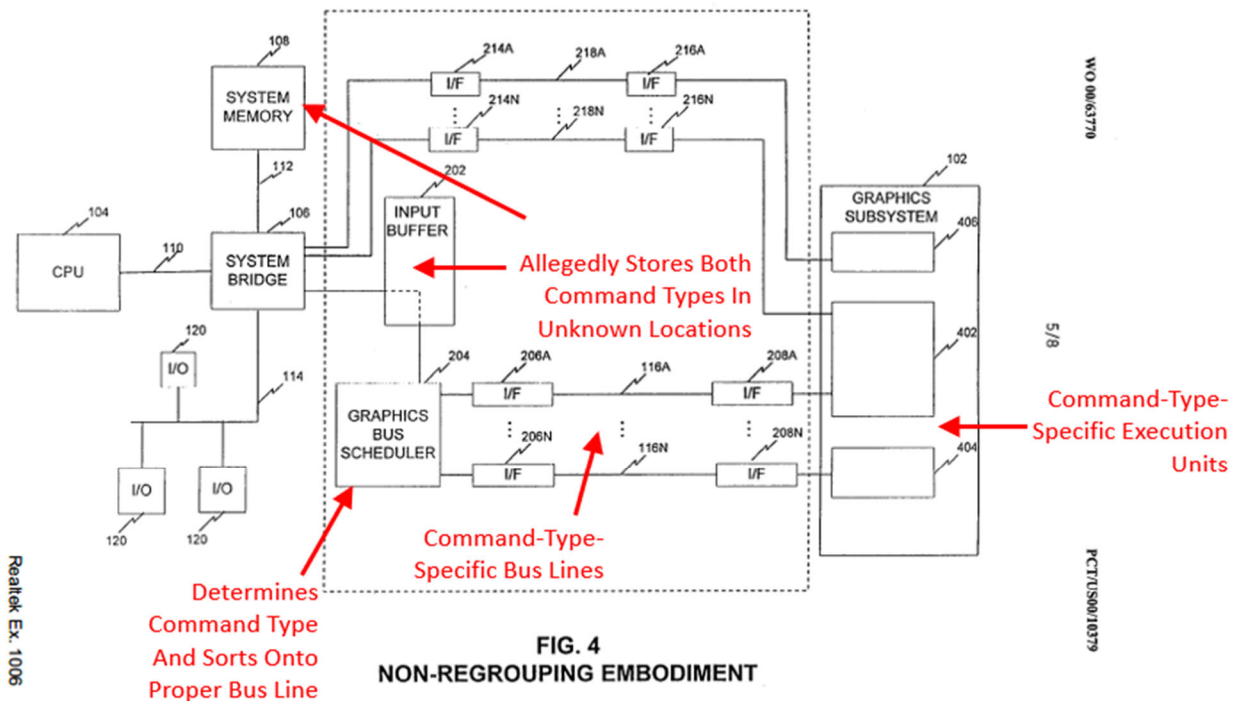
increases costs to Stuttard's system when compared to well-developed and easily obtainable off-the-shelf components such as Intel's AGP bus. Ex. 2009, ¶131. Therefore, a POSITA would not be motivated to exchange the AGP bus of Stuttard with Williams. *Id.*

Petitioner 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. Pet., 43-44; Ex. 2009, ¶132. These are generic reasons which are common to most graphics systems and not detailed enough to establish success. *Id.* Petitioner's other reason, that both involve processing of conventional graphics objects including pixel and vertex commands, is also, by Petitioner's own admission, conventional of graphics systems. *See also* Ex. 1001, 1:24-28. Based on Petitioner's reasons, a POSITA would have a reasonable expectation of success in combining almost every reference on graphics systems, which is incorrect. Ex. 2009, ¶132.

C. Ground 3: Williams In View Of Whittaker and the Knowledge Of A POSITA

Ground 3 fails to render the challenged claims invalid. Ex. 2009, ¶133. Neither reference discloses or renders obvious storing pixel and vertex command threads, as described below. *Id.*

Further, the alleged “arbiter” in this combination, Williams’s GBS, does not perform any selecting of command threads, as required by all claims. It merely receives what it receives and puts each command onto the proper bus line for that command type; nor is there any reason to modify it to do anything differently, as discussed in Ground 2:



Williams and Whittaker are also completely unrelated references. No POSITA would think to look from Williams to Whittaker for any reason, because they conflict with one another and are not remotely in the same field of technology.

As with Stuttard, Whittaker executes based on priority. *See, e.g.*, ID at 48 (“the MCC identifies the operation with the highest priority and passes that to the data processing units for execution.”). As described in Ground 2, this is fundamentally incompatible with Williams’s approach, which is to strictly maintain the order of commands as received and process things in predetermined, rigid way, with particular command types being sent on dedicated buses to dedicated processing units to be processed according to command type, not priority. The two cannot be merged as they are fundamentally different approaches; implementing priority-based reordering is not a *benefit* to Williams, it is a change that is in direct contradiction to the chosen approach. Ex. 2009, ¶¶153-155.

Further, Whittaker is about processing video, *i.e.*, images from cameras, or other real-time data streams such as audio. Ex. 1007 at 3:34-44 (“Detailed Description” stating “The base architecture . . . has a plurality of inputs and outputs which can be coupled to real time data input and output devices 4. These can be, for example, video sources, audio sources, video outputs, audio outputs, data sources, storage devices etc.”); Ex. 2009, ¶135. Williams is about 3D graphics generated by GPUs and graphics subsystems. Ex. 2009, ¶135; Ex. 1006 at 3-11. Williams contains no references to 3D graphics processing; at best it only

mentions offhand the storage of texture maps, which are not explained to be 3D graphics texture maps. *Id.* Rather, Whittaker is concerned with concepts like how the digital to analog conversion (DAC), used in video processing but not graphics processing. Ex. 1007, 6:30-56. The video processing described in Whittaker is in television sets, VCRs, DVDs, video players, video scalars and other devices. Ex. 2009, ¶135.. Those bear no relationship to concepts like GPUs that are the focus of Williams and have no relationship to transforming 3D objects into 2D screen space. *Id.*

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

As explained above in Section VI.B.2, Williams does not disclose the storage of command threads, as required in all claims. Ex. 2009, ¶134. Petitioner does not rely on Whittaker for this claim limitation. Pet., 56. Nor could it because Whittaker does not disclose pixel or vertex command threads. Ex. 2009, ¶134. While the ID identified Whittaker's discussion of processing threads according to available resources and priorities, these threads are not pixel or vertex command threads. ID, 53 (citing Ex. 1007, 4:44-61). Specifically, Whittaker at 4:44-61 describes a cache bank for storing data, such as texture maps. Ex. 2009, ¶134. The cache banks do not store threads; they store data.

Separately, Petitioner 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 VI.A.1.d.

2. All Claims: Williams’ Does Not Disclose or Render Obvious the Claimed Arbiter

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. This requirement being met by Williams is not alleged or analyzed in any narrative portion of the Petition.

In the claim chart for Ground 3, without any explanation, Petitioner identifies GBS 202 is the claimed arbiter. Pet., 56-58. However, Williams’s GBS does not *select* graphics commands (or anything). Ex. 2009, ¶139. 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. 1006, 18:26-28. 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. Ex. 2009, ¶139. Dr. Pedram agreed that any so-called arbiter which merely receives does not select, testifying: “If the so-called ‘arbiter’ had nothing to do with what inputs arrive at its input terminals, then,

of course, it had no selection of choice at that point.” Ex. 2010, 205:17-20. Therefore, GBS does not “select” according to Dr. Pedram’s testimony. 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. Ex. 1006, 18:26-27, 19:4-7 (providing that GBS receives commands prior to sorting). And the sorting cannot be the claimed selecting. Ex. 2009, ¶139. First, the claimed selecting must be done from the memories, as claimed. *Id.* Second, there is no sense of “picking out” as the patentee stated was the meaning of “select.” Ex. 1002 (’053 FH), 107. The GBS simply puts the command on the bus corresponding to its type. Ex. 2009, ¶139.

As Dr. Pedram further admits, even if that sorting could involve some sort of selection, Williams does not describe it. Ex. 2009, ¶140. 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. Ex. 2010, 309:19-310:6. But Dr. Pedram 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. Ex. 2009, ¶140.

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

Both claims 2 and 5 require a “command processing engine” that can process both pixel and vertex command threads. Section V.C. As explained above, Williams in view of Whittaker does not disclose pixel command threads or vertex command threads. *Id.* 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. *Id.*

Petitioner’s identified components, “functional components” 402 and 404, are not the claimed “command processing engines.” Pet., 58-59; Ex. 2009, ¶143. This is because they are *type-specific*, meaning that each functional component can only perform geometry operations or pixel operations, and not both. *Id.* Williams

discloses that functional components 402 and 404 “perform geometry and pixel operations, *respectively*.” Williams also states that the graphics commands can be “transferred to the appropriate component/stage of a graphics pipeline 700.” *Id.*, 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.* Thus, only dedicated, type-specific components in Williams are alleged to be the claimed command processing engines, contrary to the claims. Ex. 2009, ¶143.

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

Ground 3 does not disclose or render obvious these claims for the same reasons explained in Sections VI.B.1, VI.B.4. Ex. 2009, ¶146.

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

Petitioner identifies disclosures from Williams in a claim chart without any explanation as to why they disclose or render obvious this claim. The disclosures mention, *e.g.*, geometry calculations, curve and surface geometry, processing geometric primitives, and operations on pixels. Pet., 60 (citing Ex. 1006, 2:26-30, 24:18-23, 24:27-25:2). But none of them references the required arithmetic logic

unit. Ex. 2009, ¶147. Petitioner’s lack of explanation and conclusory statements cannot sustain a finding of invalidity for this claim. *Wasica Fin. GmbH v. Cont’l Auto. Sys.*, 853 F.3d 1272, 1286 (Fed. Cir. 2017).

6. Claim 7: Williams Does Not Disclose or Render Obvious a Texture Processing Engine

Nor does Petitioner explain how the alleged command processing engines, functional components, comprise a texture processing engine. Ex. 2009, ¶147. Petitioner solely identifies the disclosure that “graphics subsystem 102 processes texture data” without any further explanation. This disclosure does not provide where in graphics subsystem 102 texture data is processed, *e.g.*, in the claimed processing engine. Ex. 2009, ¶148. Petitioner’s alleged command processing engines are functional components 402, 404, with no disclosure of them being texture processing engines. Ex. 2009, ¶148.

7. No Motivation to Combine or Reasonable Expectation of Success

A POSITA would not have been motivated to combine Williams and Whittaker or have a reasonable success in doing so. Ex. 2009, ¶149. First, the two references are focused on different as explained above in Section VI.C. Williams is focused on 3D graphics processing whereas Whittaker is focused on video processing. *Id.* Whittaker focuses much of its discussion on video and audio input data, which is not relevant to 3D graphics processing. *See, e.g.*, Ex. 1007, 4:15:25,

5:53-60, 6:13-56, Fig. 3. 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; Ex. 2009, ¶150. Moreover, Whittaker certainly makes no reference to the processing of pixel and/or vertex instructions that are allegedly the focus of Williams. Ex. 2009, ¶¶151-152.

Further, the thread priority used by Whittaker is incompatible with Williams' ordered graphics commands. Ex. 2009, ¶¶153-155. Key to Williams's approach to speed 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. 1006, Abstract, 9:1-7, 18:14-22, 23:1-10; Ex. 2009, ¶¶153-155. In contrast, 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 reorders threads based on priority. Ex. 2009, ¶154. This is in direct conflict with Williams's ordered graphics command approach; Ex. 2009, ¶155.

Lastly, Petitioner claims that the Board should find compelling its previously allowance of Williams with Moreton. Pet., 53. But Petitioner fails to properly explain how Whittaker is analogous art to Moreton. *Id.* Thus, the allegations fail.

VII. CONCLUSION

For any of the foregoing reasons, the Board should determine that Petitioner has failed to prove that the challenged claims of the '053 Patent are unpatentable.

Dated: January 17, 2024

Respectfully submitted,

/s/ William A. Meunier

William A. Meunier (Reg. No. 41,193)

Michael T. Renaud (Reg. No. 44,299)

Adam S. Rizk (Reg. No. 66,867)

MINTZ, LEVIN, COHN, FERRIS,
GLOVSKY AND POPEO, P.C.

One Financial Center

Boston, MA 02111

Telephone: 617-542-6000

Facsimile: 617-542-2241

E-mails: WAMeunier@mintz.com;

ARizk@mintz.com

MTRenaud@mintz.com

Reza Dokhanchy (Reg. No. 62,795)

MINTZ, LEVIN, COHN, FERRIS,
GLOVSKY AND POPEO, P.C.

3580 Carmel Mountain Road, Suite 300

San Diego, CA 92130

Telephone: 858-314-1596

RDokhanchy@mintz.com

CERTIFICATE OF WORD COUNT

Pursuant to 37 C.F.R. §42.24(d), Patent Owner hereby certifies, in reliance on the word count of the word-processing system (Microsoft Office Word 2010) used to prepare this preliminary response, that the number of words in this paper is 13,923. This word count excludes the tables of contents, tables of authorities, certificate of word count, certificate of service, and table of exhibits.

Dated: January 17, 2024

/s/ William A. Meunier

William A. Meunier (Reg. No. 41,193)

CERTIFICATE OF SERVICE

I certify that a copy of Patent Owner's Response and accompanying ATI Exhibits 2009 through 2012 is being served by electronic mail on the following counsel of record:

Lead Counsel

Jeffrey Johnson
Reg. No. 53,078
ORRICK, HERRINGTON &
SUTCLIFFE, LLP
609 Main Street, 40th Floor
Houston, TX 77002-3106
Telephone: (713) 658-6400
Facsimile: (713) 658-6401
Email: 3J6PTABDocket@orrick.com;
Realtek-AMD_OHS@orrick.com

Backup Counsel

Christopher J. Higgins
Reg. No. 66,422
ORRICK, HERRINGTON &
SUTCLIFFE, LLP
1152 15th Street, N.W.
Washington, DC 20005-1706
Telephone: (202) 339-8400
Facsimile: (202) 339-8500
Email: OCHPTABDocket@orrick.com;
Realtek-AMD_OHS@orrick.com

Steve Baik
Reg. No. 42,281
WHITE HAT LEGAL
P. O. Box 382
San Jose, CA 95002
Telephone: (650) 618-5282
Email: sbaik@whitehat.legal

Dated: January 17, 2024

/s/ William A. Meunier

William A. Meunier (Reg. No. 41,193)