

**UNITED STATES PATENT AND TRADEMARK OFFICE**

---

**BEFORE THE PATENT TRIAL AND APPEAL BOARD**

---

REALTEK SEMICONDUCTOR CORP.,

and

TCL INDUSTRIES HOLDINGS CO., INC.

Petitioner

v.

ATI TECHNOLOGIES ULC

Patent Owner

---

Case No.: IPR2023-00922

U.S. Patent No. 8,760,454<sup>1</sup>

Issue Date: June 24, 2014

Title: Graphics Processing Architecture Employing a Unified Shader

---

**DECLARATION OF HANSPETER PFISTER, PH.D. IN SUPPORT OF  
PETITIONER'S REPLY TO PATENT OWNER'S RESPONSE TO  
PETITION IN THE *INTER PARTES* REVIEW OF U.S. PATENT NO.  
8,760,454**

---

<sup>1</sup> Joinder with IPR2024-00366

## TABLE OF CONTENTS

|                                                                                                                                                                                                                                                                                        | Page |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| I. INTRODUCTION .....                                                                                                                                                                                                                                                                  | 1    |
| II. CLAIMS 1-11 ARE UNPATENTABLE AS BEING OBVIOUS<br>OVER LINDHOLM '685 PATENT AND LINDHOLM '913<br>PATENT .....                                                                                                                                                                       | 1    |
| A. ATI Adds Limitations to Claims 1, 3, 4, and 5 That Are Not<br>Required .....                                                                                                                                                                                                        | 1    |
| 1. The '454 patent does not require an "evaluation" of<br>available storage space.....                                                                                                                                                                                                 | 2    |
| 2. Lindholm describes a thread prioritization scheme based<br>on available storage space to store source data. ....                                                                                                                                                                    | 5    |
| 3. Lindholm discloses a Resource Scoreboard that keeps<br>track of incoming data by monitoring the status of source<br>registers in the Register File. ....                                                                                                                            | 7    |
| 4. Lindholm explicitly discloses vertex and pixel operations<br>based on the availability of data at the input buffer.....                                                                                                                                                             | 8    |
| B. The Lindholm '685 and '913 Patents Disclose A Processor Unit<br>That Performs Vertex Manipulation Operations and Pixel<br>Manipulation Operations At Various Degrees Of Completion<br>Based On Switching Between Instructions In The Instruction<br>Store As Seen In Claim 11. .... | 9    |
| 1. Claim 11 does not require switching between<br>"unfinished" vertex and pixel operations.....                                                                                                                                                                                        | 9    |
| 2. Lindholm discloses a multithreaded unified shader,<br>including the ability to switch between pixel and vertex<br>operations. ....                                                                                                                                                  | 12   |
| 3. Lindholm discloses an embodiment of switching between<br>operations of different types at various degrees of<br>completion.....                                                                                                                                                     | 15   |
| C. The Lindholm '685 and '913 Patents Disclose Execution of<br>Instructions "In Response to" Receiving Selected Data (Claim<br>2).....                                                                                                                                                 | 17   |

**TABLE OF CONTENTS**  
(continued)

|                                                                                                                                                                               | <b>Page</b> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| 1. ATIs interpretation of “in response to selected data from the general purpose register” is not supported by the evidence and goes counter to common usage by a POSITA..... | 18          |
| 2. Lindholm demonstrates the execution of operations “in response to selected data from the general purpose register” (Claim 2). ....                                         | 22          |
| 3. ATIs distinction between “data during processing” vs. “data that is to be processed” is not supported by the evidence. ....                                                | 25          |
| D. The Lindholm ’685 and ’913 Patents Disclose “Memory” Separate From the “Store” (Claim 6).....                                                                              | 26          |
| 1. Lindholm discloses a sequencer connected to “memory” that is separate from the “store.”.....                                                                               | 26          |
| E. The Lindholm ’685 and ’913 Patents Disclose a “Control Signal” (Claims 7, 10) and “Arbiter” (Claim 10) .....                                                               | 30          |
| 1. “Thread entries,” “state data,” or “priority” are control signals.....                                                                                                     | 30          |
| 2. Lindholm discloses the claimed “control signal” and “arbiter.”.....                                                                                                        | 31          |
| 3. ATI is confused by the common understanding of the term “information.”.....                                                                                                | 33          |
| III. Ground #2: Claims 1-11 based on the Combination of Amanatides and Kohn.....                                                                                              | 35          |
| A. The Amantides + Kohn Patents Determine Which Data To Process By Evaluating Storage Capacity As Seen In Claims 1, 3, 4, and 5. ....                                         | 35          |
| 1. The ‘454 patent does not require an “evaluation” of available storage space.....                                                                                           | 35          |
| 2. Amanatides + Kohn disclose a priority scheme for processing pixel and vertex data that is based on storage capacity. ....                                                  | 36          |

## TABLE OF CONTENTS

(continued)

|                                                                                                                                                                                                                                                             | Page |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 3. Amanatides + Kohn disclose processing of pixel and vertex data based on the availability of shared resources.....                                                                                                                                        | 39   |
| 4. The ID correctly identified the priority scheme in Amanatides + Kohn as disclosing processing of pixel and vertex data based on the availability of shared resources.....                                                                                | 41   |
| 5. ATI falsely imbues intelligent behavior to the graphics processing system of Amanatides.....                                                                                                                                                             | 45   |
| B. The Amantides + Kohn Patents Disclose A Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11). .... | 45   |
| 1. Kohn discloses operations “at various degrees of completion based on switching between 30 instructions in the instruction store.” .....                                                                                                                  | 46   |
| 2. A POSITA would be motivated to combine Amanatides + Kohn.....                                                                                                                                                                                            | 48   |
| C. Amantides + Kohn Disclose “Selected Data” (claims 2 and 5).....                                                                                                                                                                                          | 49   |
| 1. Amanatides does not impose any temporal restrictions of the processing of GM and RM packets.....                                                                                                                                                         | 50   |
| 2. Reading data from FIFOs in Amanatides is “selecting data” from a data store. ....                                                                                                                                                                        | 52   |
| 3. Claims 7 and 10 do not require that the “Selection Circuit” is within the processor of the “Unified Shader”. ....                                                                                                                                        | 54   |
| 4. Amanatides and Kohn include the “Selection Circuit” within the “Unified Shader” as required by claims 7 and 10.....                                                                                                                                      | 55   |
| D. Amantides + Kohn Disclose a “Control Signal” (claims 7 and 10).....                                                                                                                                                                                      | 57   |
| 1. Packets in Amanatides and Kohn include a priority.....                                                                                                                                                                                                   | 57   |
| 2. ATI is relies on colloquial sources and non-technical terminology. ....                                                                                                                                                                                  | 59   |

**TABLE OF CONTENTS**  
(continued)

|                                                                                                                                                                                                                                               | <b>Page</b> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| E. Amantides + Kohn Disclose a “Selection Circuit” and “Arbiter,” (claim 10) .....                                                                                                                                                            | 60          |
| 1. ATI fails to understand the basic circuits of FIFOs and buses.....                                                                                                                                                                         | 60          |
| 2. Packets in Amanatides are sent from FIFOs over the bus to DRAM, VRAM, and the on-chip cache. ....                                                                                                                                          | 62          |
| 3. The “control signal” in Amanatides prompts information to be transmitted and stored. ....                                                                                                                                                  | 64          |
| 4. Claim 10 does not require specific timing for arbitration.....                                                                                                                                                                             | 65          |
| IV. Ground #3: Claims 1-11 based on the Combination of Selzer and Fiske.....                                                                                                                                                                  | 67          |
| A. The Selzer + Fiske Patents Determine Which Data To Process By Evaluating Storage Capacity As Seen In Claims 1, 3, 4, and 5. ....                                                                                                           | 67          |
| 1. The ‘454 patent does not require an “evaluation” of available storage space.....                                                                                                                                                           | 67          |
| 2. Selzer discloses processing of pixel and vertex data based on storage capacity.....                                                                                                                                                        | 67          |
| 3. Fiske’s hardware context switch discloses processing of pixel and vertex data based on storage capacity. ....                                                                                                                              | 71          |
| 4. Claim 1 in the ‘454 patent does not impose timing constraints or ordering on the processing of vertex and pixel data. ....                                                                                                                 | 73          |
| 5. Selzer + Fiske do not impose any timing constraints on the processing of vertex and pixel data.....                                                                                                                                        | 76          |
| 6. Selzer + Fiske discloses performing pixel calculation operations and then vertex calculation operations based on resources and vice versa (claims 3 + 4). ....                                                                             | 78          |
| B. Selzer + Fiske Disclose A Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11). .... | 81          |

## TABLE OF CONTENTS

(continued)

|                                                                                                                                                                                        | Page |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 1. Fiske discloses performing operations “at various degrees of completion based on switching between instructions in the instruction store.” .....                                    | 82   |
| 2. The combination of Selzer and Fiske discloses claim 11.....                                                                                                                         | 83   |
| 3. There is motivation to combine Selzer and Fiske for fast multithreaded graphics processing.....                                                                                     | 84   |
| 4. Selzer + Fiske disclose switching between incomplete instruction in the instruction store. ....                                                                                     | 85   |
| C. Selzer + Fiske Disclose Execution of Instructions “In Response to” Receiving Selected Data (Claim 2). ....                                                                          | 87   |
| 1. ATIs interpretation of “in response to selected data from the general purpose register” is not supported by the evidence and goes counter to common usage by a POSITA.....          | 87   |
| 2. Selzer demonstrates the execution of operations “in response to selected data from the general purpose register” (Claim 2). ....                                                    | 87   |
| 3. Fiske demonstrates the execution of operations “in response to selected data from the general purpose register” (Claim 2). ....                                                     | 90   |
| D. Selzer + Fiske Disclose the “Sequencer,” “Instruction Store,” “Circuitry,” “Selection Circuit,” and “Arbiter” Within the “Unified Shader” As Required (Claims 2, 5-7, 10, 11). .... | 91   |
| 1. Selzer’s “Master Module” discloses a “Sequencer” and “Circuitry” (claims 2, 5, and 6).....                                                                                          | 92   |
| 2. Selzer + Fiske Disclose a “Selection Circuit” and “Arbiter” (claims 7 and 10).....                                                                                                  | 95   |
| E. The “Sequencer” And “Instruction Store” in Selzer Maintain Instructions (Claims 2, 5, and 11) .....                                                                                 | 96   |
| F. Selzer + Fiske Disclose the “Arbiter” (Claim 10) and “Control Signal” (Claims 7, 10) .....                                                                                          | 98   |
| G. Motivation to Combine Selzer + Fiske .....                                                                                                                                          | 99   |

**TABLE OF CONTENTS**  
(continued)

|                                                                                                                                       | <b>Page</b> |
|---------------------------------------------------------------------------------------------------------------------------------------|-------------|
| 1. Both Fiske and Selzer adapt the activities of the processing units to the actual processing requirements.....                      | 99          |
| 2. The multithreading in Fiske improves the performance of rendering calculations in Selzer.....                                      | 100         |
| 3. Task switching in Selzer is not the same as multithreading in Fiske, but it helps motivate the combination of Selzer + Fiske ..... | 102         |
| 4. Combining Selzer and Fiske would be obvious and easy for a POSITA. ....                                                            | 104         |
| H. Secondary Considerations .....                                                                                                     | 105         |
| V. DECLARATION.....                                                                                                                   | 107         |

## **I. INTRODUCTION**

1. I was asked to provide this declaration containing my rebuttal opinions and analysis to issues raised in Dr. Mangione-Smith's declaration dated February 23, 2024 and the corresponding Patent Owner Response (POR or Response) to which it alleges to support. Having carefully read and evaluated all of the issues raised in those documents, it is clear that my opinions are confirmed and claims 1-11 of the '454 patent are unpatentable.

## **II. CLAIMS 1-11 ARE UNPATENTABLE AS BEING OBVIOUS OVER LINDHOLM '685 PATENT AND LINDHOLM '913 PATENT**

### **A. ATI Adds Limitations to Claims 1, 3, 4, and 5 That Are Not Required**

2. ATI argues that Lindholm does not disclose "Executing Operations Depending Upon an Amount Of Space Available In The Store (Claim 5), Much Less Performing Vertex Operations Or Pixel Operations Until Enough Storage Is Available For The Other Operation Type (Claims 1, 3, 4)" Response Sec. 2, p. 29.

3. In support, ATI argues that the Lindholm patents are not "evaluating available storage space for incoming data, much less doing so to determine what to execute." Response p. 31. As shown in the Petition and below, ATI is incorrect on every point.

**1. The ‘454 patent does not require an “evaluation” of available storage space.**

4. In their response, ATI incorrectly use the term “evaluating” available storage space, which does not appear in the ‘454 patent. “The ‘454 Patent’s Unified Shader Determines Which Data To Process By Evaluating Storage Capacity As Seen In Claims 1, 3-10”, Sec. III A. The ‘454 patent claims and description do not use the terms ‘evaluating’ or ‘evaluation’ or ‘determine’ or ‘determination’ in conjunction with storage space. Instead, the claims require the ability to store data in registers that are free or available. There is no need for any “evaluating” of free storage space, nor do the claims and patent description of the ‘454 patent mention any “evaluating” or “evaluation” of storage space. For example, a first-in, first-out (FIFO) memory allows new data to be stored without first “evaluating” whether there is available space. If the FIFO is full, i.e., has no more free storage space, its circuitry overwrites the oldest entry in its registers with new incoming data. The embodiment of storage space as a FIFO is entirely within the scope and claims of the ‘454 patent. To the extent any evaluation steps is done with respect to the available space, it is done by the arbiter, which is not part of the claimed unified shader in the ‘454 patent. As shown in Figure 4A and described in the ‘454 patent specification, the arbiter is a stand-alone component at least one component (MUX) removed from the unified shader. However, all of the ‘454

patent claims are directed specifically to the unified shader, e.g., claim 1 – “A method carried out by a unified shader comprising,” claim 2-5, 11 – “A unified shader comprising.” Further, the only “determination” made based on that “evaluation” or “consideration” of available space is also done by the arbiter when it determines what vertex or pixel data to transmit to the general purpose register block 92 via the MUX 66. Ex. 1001, 5:41-44. The unified shader of the ’454 patent has no such evaluation or determination functionality and, instead, merely operates on the information passed to the general purpose register block via the multiplexer.

5. In my opinion, the argument by ATI that the claims require “considerations for determining what to process” is false. Response p. 31 a POSITA would not understand there to be any “determining” or “determination” in the ’454 patent concerning what to process or concerning storage space carried out by the claimed unified shader. Similarly, ATI’s statement that Lindholm does not “disclose the required evaluation of available storage space for incoming data to determine what to execute” also reads false requirements into the claims. “Evaluation” of available storage space is not required by the unified shader, especially not in conjunction with determining what to execute.

6. Claim 1 of the ‘454 patent requires performing vertex operations “unless the general purpose register block does not have enough available space therein to store incoming vertex data.” It also requires continuing pixel calculation operations “until enough registers within the general purpose register block become available.” The patent claim refers to the availability of storage space and does not require an “evaluation” of the storage space.

7. Claim 3 of the ‘454 patent requires performing pixel calculation operations “until enough shared resources become available and then use the shared resources to perform vertex calculation operations.” No “evaluation” of shared resources is required in the claim. In addition, shared resources could refer to storage, arithmetic-logic units (ALUs), or other computational units.

8. Claim 4 of the ‘454 patent requires the execution of vertex calculation operations “until enough shared resources become available and then use the shared resources to perform pixel calculation operations.” No “evaluation” of shared resources is required in the claim. In addition, shared resources could refer to storage, arithmetic-logic units (ALUs), or other computational units.

9. Claim 5 of the ‘454 patent requires the execution of pixel and vertex calculation operations “depending upon an amount of space available in the store,” where the term “store” refers to data storage. No “evaluation” of storage space is

required in the claim. Thus, a POSITA would not consider or understand these claims to require any “evaluation” or “determination” steps that ATI is attempting to create.

**2. Lindholm describes a thread prioritization scheme based on available storage space to store source data.**

10. ATI claims: “For example, prioritizing based on how much data is in the buffers doesn’t disclose the requirement of claim 1 to perform vertex operations until there is no space for new incoming vertex data, in which case performing only pixel calculations until there is once again room for new vertex data.” Response at p. 32. In my opinion, ATI is wrong for several reasons. Performing vertex or pixel operations based on the available storage space is a form of prioritization. Vertex or pixel operations get priority, i.e., executed, depending on the available storage space.

11. Lindholm discloses such a prioritization scheme based on available storage space, including storage in the vertex and pixel input buffers. The ID correctly found that this element is met because “Lindholm ’685 discloses determining thread execution priority based on the amount of sample data in the pixel and vertex input buffers and recognizes that the maximum number of threads that can be executed simultaneously depends on the size of storage for thread state data.” ID, 40-41.

12. Moreover, even taking ATI's wrong limitations into account, Lindholm discloses all of the elements in claims 1, 3, 4, and 5. The '685 patent uses a complex arbitration scheme in the Instruction Cache and Instruction Scheduler in order to select which instructions from the available threads will be processed based on the available data. These units can select "any instruction within the IWU 435 [to] be executed based on the availability of computation resources in an Execution Unit 470 and ***source data stored in Register File 350.***" *Ex. 1005* at 7:56-64; 8:47-64. (emphasis added) Thus, the '685 patent employs arbitration at the data level based on the "availabil of "source data stored in Register File 350." This indicates to a POSITA that there must be available storage space in Register File 350 to store source data. In other words, it is obvious that the instruction is only being executed if there is space to store source data.

13. If a particular program instruction specifies available source data, the scheduler will schedule that program instruction to be executed, because doing so would not stall the system. *Id.* at 8:49-53. ***If the specified source data is not available***, the scheduler will instead schedule a different thread's program instruction for execution. *See id.* (emphasis added) In short, the scheduler bases its decision to execute a particular program instruction based on the availability of source data specified by that instruction. *Id.* at 8:49-53; 17:41-44. Program

execution is conducted on an instruction-by-instruction basis, and the scheduler switches at will between instructions and threads of various sample types. *Id.* at 17:41-44.

**3. Lindholm discloses a Resource Scoreboard that keeps track of incoming data by monitoring the status of source registers in the Register File.**

14. Lindholm also disclosed a way to track storage space using a Resource Scoreboard. The Resource Scoreboard tracks the status of source registers scheduled to be written during processing (*i.e.*, destination registers). These registers are marked as “write pending.” When a write operation is completed, and the data in the destination register becomes available for use as source data, the Resource Scoreboard indicates this change. Ex. 1005, 8:58-65 In some embodiments, the Resource Scoreboard snoops (*i.e.*, observes) the interface between the Execution Unit 470 and Register File 350 to monitor incoming data. *Id.*, 9:53-56

15. The Resource Scoreboard uses the location of each program instruction’s required source data to track whether that data is available for processing vertex or pixel threads, as well as whether the necessary computational resources are available. *Id.* at 8:58-60; 9:2-4, 16:53-54, 17:45-46. The scheduler uses this information from the resource scoreboard to determine which program

instructions have their source data available, and consequently, which program instructions to schedule for execution. *See, e.g.*, Ex. 1003 at 14:20-30.

16. Once the program instructions have been scheduled for execution, a third component, the instruction dispatcher, gathers the specified source data and scheduled instructions and outputs them to the execution unit so that the corresponding sample may be processed. *Id.* at 9:40-42; 16:55-57; 17:47-50. This completes the processing operation.

**4. Lindholm explicitly discloses vertex and pixel operations based on the availability of data at the input buffer.**

17. ATI claims that Lindholm does not disclose “evaluating available storage space for incoming data, much less doing so to determine what to execute.” Response at p. 31. This is false, as can be seen with direct quotes from the ‘685 patent: “Each Execution Pipeline 240 signals to Vertex Input Buffer 220 when a sample can be accepted or when a sample cannot be accepted.” Ex. 1005 at 4:34-36 “Each Execution Pipeline 240 signals to Pixel Input Buffer 240 when a sample can be accepted or when a sample cannot be accepted as described further herein.” Ex. 1005 at 5:24-26.

18. Accordingly, the undisputed evidence establishes that Lindholm discloses executing operations depending upon the amount of space available in the store (Claim 5) and performing vertex operations or pixel operations until

enough storage is available for the other operation type (Claims 1, 3, 4). At a minimum, all of these disclosures render these claims obvious.

**B. The Lindholm '685 and '913 Patents Disclose A Processor Unit That Performs Vertex Manipulation Operations and Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store As Seen In Claim 11.**

19. Petitioner and ATI agree that Lindholm discloses a unified shader (processor unit) to execute pixel or vertex operations. However, ATI wrongly claims that Lindholm does not disclose that the processor can switch between “unfinished vertex operations and pixel operations,” as required by claim 11. In my opinion, this is false.

**1. Claim 11 does not require switching between “unfinished” vertex and pixel operations.**

20. First, the '454 patent and claim 11 do not specify the timing of vertex and pixel operations. They could happen sequentially, without overlap in time, or simultaneously, at the same time. Switching between pixel or vertex operations could occur while these operations are executed sequentially or simultaneously. ATI does not dispute and agrees with the Petitioner that Lindholm discloses a processor capable of switching between pixel and vertex operations based on instructions in the instruction store.

21. Claim 11 requires that the vertex and pixel operations related to the instructions are “at various degrees of completion.” This includes operations that have just started, are in process, or have finished. It is not, as ATI claims, only a “switch between *unfinished* vertex operations and pixel operations.” Response p. 32. (emphasis added) The ATI term “unfinished,” which excludes switching between completed operations, differs from the claim term “at various degrees of completion,” which includes switching between finished operations.

22. ATI incorrectly states that I agreed this claim “requires the processor unit to switch between unfinished vertex operations and pixel operations.” Response 32. The question asked of me was an unrealistic and incomplete hypothetical in which I had to make several assumptions to try and answer. Initially I was asked to assume “full completion,” which was a fuzzy concept, which I indicated was not a concept I had considered. Ex. 2123, 60:5-61:1. Mr. Dokyanchy implied I should assume that “full completion” and “completion” (which is what is recited in the claim) were the same by stating he “was just using the claim language.” *Id.*, 61:5-6. I do not agree that “completion” in the claim means “full completion.” I was then asked to assume “there are definite boundaries of when an operation begins and when an operation ends.” *Id.* 61:7-10. Next, I was asked to assume that two “operations” (1 and 2) had a “defined

boundary.” *Id.*, 61:13-14. It was not clear if the “defined boundary” for each operation was meant to include the previous assumption of beginning and end. No other details were provided about whatever assumed system or unified shader might be carrying out the hypothetical operations. Then I was asked to answer the questions “would it meet the claim language if the switch between operation 1 and operation 2 was after operation 1 was completed?” *Id.*, 61:16-18. No details were provided about switching, what switching meant, if I was supposed to assume a switch as claimed, how many other switches are made, etc., other than the question. The question implies a single switch between operation 1 and 2. I assumed there must be at least one other switch because the claim requires “various” degrees of completion, which to a POSITA means there must be at least two. The only switch I was given to assume in the question was a switch after operation 1 was complete. *Id.*, 61:16-18. Because after completion of operation 1 the only operation left is operation 2, there could never be a second switch. So, to satisfy the claim language, if there are only two operations, the only way to get “various degrees of completion” would be to have a switch before completing operation 1. That would leave some operations left (e.g., to come back and finish operation 1, or to execute some or all of operation 2) where a second switch could take place to satisfy the “various” and/or plural requirement of the claims.

However, if you have more than two operations, you can have switches where the switch occurs after completion and still satisfy the claim language. Furthermore, if these operations do not have to be finished, as was assumed in the question by adding that “completion” implied “full completion” to the hypothetical, then the operations could have just started, be in process, or have finished. Nothing in claim 11 or the ‘454 patent requires switching between “unfinished” or “fully completed” vertex and pixel operations. All possible timings of vertex and pixel operations are within the limitations of claim 11 as long as they are “at various degrees of completion.”

**2. Lindholm discloses a multithreaded unified shader, including the ability to switch between pixel and vertex operations.**

23. ATI does not dispute that Lindholm discloses a unified shader. Lindholm further discloses that the unified shader uses multithreaded processing. “A graphics processor for multithreaded execution of program instructions associated with threads to process at least two sample types...” Ex. 1005 at Summary. “A graphics processor for simultaneous multithreaded execution of program instructions associated with threads to process at least two sample types including dynamic load balancing of sample types among the threads...” *Id.*, at Claim 1.

24. The multithreaded processing unit 400 may include many threads, each capable of being used to process either vertices or pixels. *Id.* at 6:52-59. Each multithreaded processing unit accepts samples for processing when a thread is available, sending a signal to the Vertex Input Buffer or to the Pixel Input Buffer when a vertex or a pixel can be accepted for processing. *Id.* at 4:31-36, 5:23-25.

25. The Lindholm multithreaded processor includes multiple thread state data. Switching between threads simply means selecting a different hardware thread for execution. Each multithreaded processing unit can simultaneously process vertex data and pixel data. “FIG. 4 is an illustration of an alternate embodiment of Execution Pipeline 240 containing at least one Multithreaded Processing Unit 400. Thread Control Unit 420 includes a TSR 325 to retain thread state data. In one embodiment TSR 325 stores thread state data for each of at least two thread types, where the at least two thread types may include pixel, primitive, and vertex ... thread state data for a thread assigned to a sample may include the sample type, e.g., pixel, vertex ... *Id.* at 6:38-49. Ex. 1006 at 7:19-42.

“Multithreaded execution of programs is possible because each thread may be executed independent of other threads, regardless of whether the other threads are executing the same program or a different program.” Ex. 1005 at 6:11-14

26. With the multithreaded parallel processing, Lindholm's processor can quickly switch between vertex and pixel operations by fetching instructions from the instruction cache. "Likewise, Thread Control Unit 420 provides pointers to shader program instructions to Instruction Cache 410. Instruction Cache 410 reads the thread state data for the thread from Thread Control Unit 420 and outputs program instructions to Instruction Scheduler 430. Instruction Scheduler 430 determines resources for processing the program instructions are available and outputs the program instructions to Instruction Dispatcher 440. Instruction Dispatcher 440 gathers any source data specified by the instructions and dispatches the source data and the instructions to Execution Unit 470 for execution." Ex. 1005 at 11:4-14.

27. The evidence clearly shows that Lindholm discloses a multithreaded unified shader, including the ability to switch between pixel and vertex operations. Indeed, using Lindholm's multithreaded parallel processing, its processor can quickly switch between vertex and pixel operations by fetching instructions from the instruction cache. Ex. 1005 at 11:4-14. This satisfies claim 11 of the '454 patent.

**3. Lindholm discloses an embodiment of switching between operations of different types at various degrees of completion.**

28. Lindholm explicitly discloses an exemplary embodiment of switching between operations at various degrees of completion. Figs. 7A and B (reproduced below) show sequential or simultaneous execution of different program instructions (“Execute First Program Instruction” 720 and 765, “Execute Second Program Instruction” 725 and 770) based on different sample (data) types (“Allocating threads to a first sample type” 710 and 755, “Allocating threads to a second sample type” 715 and 760). It is evident to a POSITA that executing first and second program instructions implies operations on first and second sample types. In other words, operations on the first sample type (e.g., vertex data) occur sequentially or simultaneously with operations on the second sample type (e.g., pixel data). This fulfills the requirement of “operations at various degrees of completion” in claim 11.

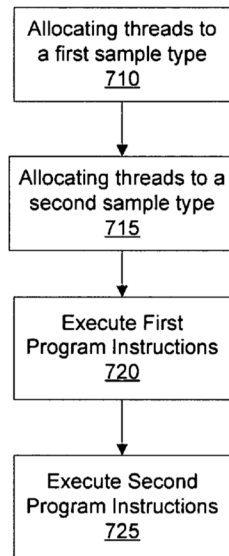


**FIG. 7A**

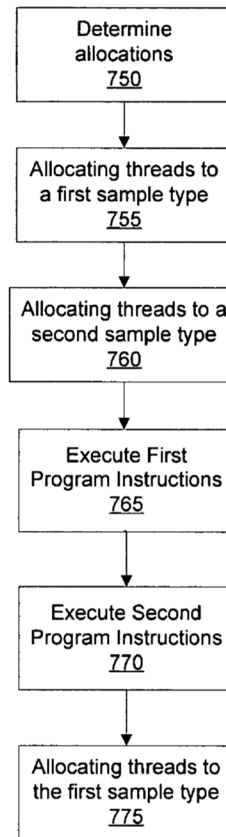


**FIG. 7B**

29. Furthermore, Fig. 7B shows an example where another thread (i.e., set of instructions) is allocated to the first sample type in step 775. Lindholm describes that this third thread of instructions and the respective operations on sample type 1 (e.g., vertex data) is at various degrees of completion regarding rendering a scene, a portion of a scene, or a new scene. “In step 775 a third number of threads determined for allocation to a first sample type are allocated to the first sample type. The third number may be allocated prior to rendering an object within a

scene, a portion of a scene, a new scene or the like. Alternatively, the third number of threads may be allocated to a third sample type.” Ex. 1005 at 13:11-17.

30. Accordingly, in my opinion, the evidence establishes that Lindholm discloses a processor unit that performs vertex manipulation operations and pixel manipulation operations at various degrees of completion based on switching between instructions in the instruction store (claim 11). At the very least, it would have been obvious to a POSITA.

**C. The Lindholm ’685 and ’913 Patents Disclose Execution of Instructions “In Response to” Receiving Selected Data (Claim 2)**

31. ATI argues that Claim 2 in the ’454 patent requires a unified shader processor that executes certain pixel or vertex thread instructions “in response to” receiving selected data from a data storage. However, in my opinion, this interpretation of the claim is wrong. Even if it were correct, Lindholm discloses execution of instructions “in response to” receiving selected data.

1. **ATIs interpretation of “in response to selected data from the general purpose register” is not supported by the evidence and goes counter to common usage by a POSITA.**

32. A POSITA understands that each GPU programmable processing core has, at a minimum, the basic components shown in the following diagram.



33. First, there must be an instruction fetch unit that retrieves the instructions that specify what needs to be processed. Second, there must be registers to store the program’s current data. Read registers are used to store data that is being fetched from outside the processing core, whereas write registers store data that has been processed and will be written back to external memory. Finally, there must be one or more Arithmetic Logical Units (ALUs) that perform the necessary arithmetic and logic operations.

34. Nothing in the ‘454 patent indicates that the supposedly invented unified shader processor deviates from this fundamental design. Fig. 5 in the patent

shows the unified shader with several source registers (93, 95, 97) corresponding to the Register Read block in the previous diagram; a general purpose register block 92 corresponding to the Register Read and Register Write blocks in the previous diagram; a “processor (e.g., CPU)” 96 corresponding to the ALUs in the previous diagram; and a sequencer 99 including an instruction store 98 corresponding to the Instruction Fetch block in the previous diagram.

35. A POSITA would understand the common interpretation of the term “in response to selected data” (claim 2) in the context of this processor design. It means that the processing units (e.g., one or more ALUs) are processing selected data stored (or maintained) in the processor's registers. This is precisely what the third clause of claim 2 in the ‘454 patent describes: “a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to *execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block*; and” ‘685 claim 2 (emphasis added)

36. The following clause in claim 2 replaces “maintained in” with “in response to”: “wherein the processor unit *executes instructions that generate a pixel color in response to selected data from the general purpose register block* and generates vertex position and appearance data *in response to selected data*

**from the general purpose register block.**” ‘685 claim 2 (emphasis added). It is clear to a POSITA that the phrase “in response to selected data from the general purpose register block” means selected data that is stored in the general purpose register block.

37. Besides the claims, the ’454 patent uses the phrase “in response” only once when describing that an arbiter circuit (not a component of the unified shader) will select an input to the multiplexor and eventually the general purpose register block in response to a control signal. Ex. 1001 at 2:62-65. Later in the patent, it is described that the control signal is sent from the arbiter to the multiplexor. *Id.*, 4:18-19. No other control signal is disclosed with respect to the arbiter circuit. Thus, these portions are irrelevant to the “in response to” interpretation because they have nothing to do with the processor unit.

38. ATI uses the term “in response to” to mean that “receipt of data causes execution of instructions” (ATI, p. 34). However, POSITA understands that this explanation is backward from what happens in a processor. Instructions are fetched and then executed based on the availability of data stored in the registers (see previous diagram). No processor would start the execution of instructions based on “receipt of data,” which is an ill-defined and fuzzy concept. If ATI were

correct, the data would have to indicate which instructions would be executed next, which is impossible as a POSITA would understand.

39. Neither the ATI nor the ‘454 patent explain how “receipt of data causes execution of instructions” or how instructions are executed “in response to” selected data from the general purpose register block. On the contrary, the patent describes how the sequencer indicates the next instruction to execute, which then causes data to be processed. The patent also describes an embodiment that “includes an arbiter circuit for selecting one of a plurality of inputs for processing *in response to* a control signal.” ‘685 2:64-66. (emphasis added) Here, in contrast to ATI's claim, a selection of inputs happens “in response” to a control signal, not “receipt of data.” Also, the control signal does not actually “cause the execution of data,” as ATI wrongly claims. It merely indicates the presence of data “for processing in response to” the signal.

40. ATI's interpretation of the meaning of “in response to” is not supported and goes counter to the understanding of a POSITA. The evidence and the patent show that “in response to” merely indicates that selected data “maintained in the general purpose register block” is ready for processing.

41. A POSITA would have understood Lindholm to render the claim obvious.

**2. Lindholm demonstrates the execution of operations “in response to selected data from the general purpose register” (Claim 2).**

42. Lindholm discloses execution of “vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block” and “instructions that generate a pixel color in response to selected data.” ‘454 claim 2 For example, when the “Thread Control Unit 320 gathers source data specified by the program instructions and dispatches the source data and program instructions to at least one PCU 375,” the PCU in response to receiving the source data and program instructions “performs computations specified by the program instructions and outputs data” to the Pixel Output Buffer or Vertex Output Buffer. ‘685 at 5:36-49. See also *id.* at 9:33-49.

43. ATIs claim that Lindholm does not “indicate that receipt of data causes execution of instructions” is based on a wrong understanding of how processors work (see Sec. 1 above). Even if there were such a mechanism, Lindholm does disclose a temporal order of first accepting data from an input buffer (“when a sample can be accepted” from the Pixel Input Buffer 240), followed by “[p]rogram instructions configure programmable computation units (PCUs) within an Execution Pipeline 240 to perform operations [on the pixel data] such as tessellation, perspective correction, texture mapping, shading, blending,

and the like,” after which the “processed samples” are output to the Pixel Output Buffer. ‘685 at 5:23-35.

44. Furthermore, the flow diagram in Fig. 8A and 8B in the ‘685 patent clearly demonstrate receiving data followed by assigning and executing threads (instructions), as shown in the following figure.

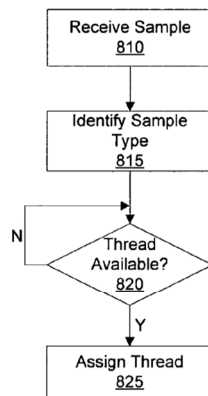


FIG. 8A

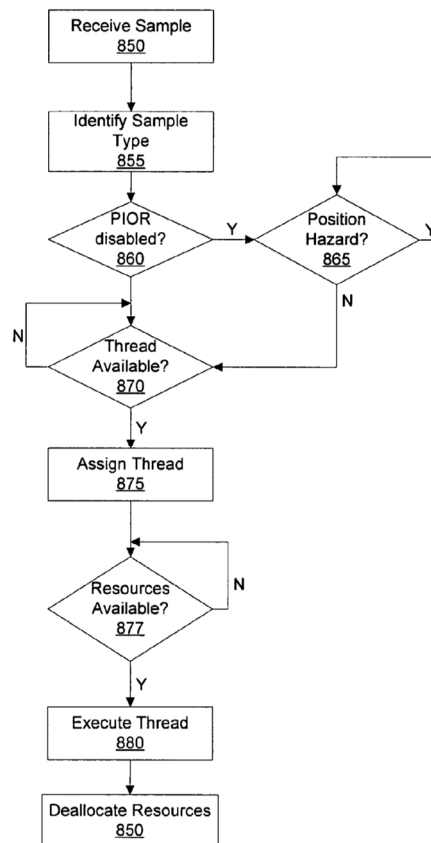


FIG. 8B

45. Fig. 8A shows in step 810 how “Thread Control Unit 320 or 420 receives a sample. In step 815 Thread Control Unit 320 or 420 identifies a sample type, e.g., vertex, pixel or primitive, associated with the sample received in step

810.” ‘685 13:20-24. In step 825, “a thread is assigned to the sample by Thread Control Unit 320 or 420. When a thread is not 35 available in step 820, Thread Control Unit 320 or 420 does not proceed to step 825 until a thread becomes available.” ‘685 13:34-37

46. Similarly, Fig. 8B shows “in step 850 Thread Control Unit 320 or 420 receives a sample. In step 855 Thread Control Unit 320 or 420 identifies a sample type, e.g., vertex, pixel or primitive, associated with the sample received in step 810.” ‘685 14:41-44 It then determines that “If in step 870 a thread is available to process the sample, in step 875 a thread is assigned to the sample by Thread Control Unit 320 or 420.” ‘685 14:62-64. “In step 880 Thread Control Unit 320 dispatches the thread assigned to the sample and source data to at least one PCU 375.” ‘685 15:14-16 Lindholm then describes how the thread instructions are executed (880). ‘685 15:16-35.

47. Fig. 8A and 8B in Lindholm demonstrate how the TCU is capable to “cause the processor unit to execute vertex calculation and pixel calculation operations on selected data” and “executes instructions that generate a pixel color in response to selected data” and “generates vertex position and appearance data in response to selected data.” (‘454 claim 2) This refutes ATIs wrong claim to the contrary.

**3. ATIs distinction between “data during processing” vs. “data that is to be processed” is not supported by the evidence.**

48. ATI seems to read a temporal aspect into claim 2 that is not supported by the ‘454 patent. ATI writes: “Thus, the space allocated in Register File 350 is for data during thread processing, not for data that is to be processed, which is what claim 2 requires.” ATI, p. 34. This presumes that claim 2 or the ‘454 patent considers the difference in time between processing and storing data. However, nothing in the claim nor the ‘454 patent supports this assertion. ATI does not elaborate or explain this claim or support it with any evidence.

49. Claim 2 in the ‘454 patent, or any of the other claims, does not mention the term time, nor does it indicate any temporal ordering between data that has been processed or data that is to be processed. Nor do the ‘454 claims speak to any difference between data that has been stored or that is to be stored. Temporal ordering or any aspect of timing is not mentioned in the ‘454 patent or its claims. The ATI statement is not supported by the patent and is, therefore, invalid.

50. Accordingly, the undisputed evidence establishes that Lindholm discloses execution of instructions “in response to” receiving selected data (Claim 2).

**D. The Lindholm '685 and '913 Patents Disclose “Memory” Separate From the “Store” (Claim 6)**

51. ATI argues that Lindholm does not disclose the following: “Claim 6, dependent on claim 5, requires “the sequencer further includes circuitry operative to fetch data from a memory.” ... The claimed “memory” must be different from the claimed “store” of claim 5.” While it is my opinion that a POSITA would not understand the memory must be separate from the store, Lindholm nevertheless discloses memory that is separate from the store as required by claim 6.

**1. Lindholm discloses a sequencer connected to “memory” that is separate from the “store.”**

52. ATI does not dispute that Lindholm discloses the claimed “sequencer.” As described in my declaration, it corresponds to “the combination of the Instruction Cache 410, the Instruction Scheduler 430 (which includes Sequencer 425 and IWU 435), and the Instruction Dispatcher 440.” Ex. 1003, ¶101. As discussed here, a POSITA would understand that Lindholm’s sequencer can communicate with buffers 215, 220, Texture Cache 230, and/or local memory 140 (where instruction dispatcher 440 fetches the data), any of which would satisfy the claimed “memory” under ATI’s incorrect interpretation.

53. As shown below, Fig. 4 in Lindholm shows the sequencer in the context of the processing unit (highlights added). The “sequencer” of claim 5 in the

‘454 patent is highlighted in yellow. It is connected to Register File 350 (“the store” in claim 5) and has connections 225, 215, and 220 external to the Execution Pipeline 240.

54.

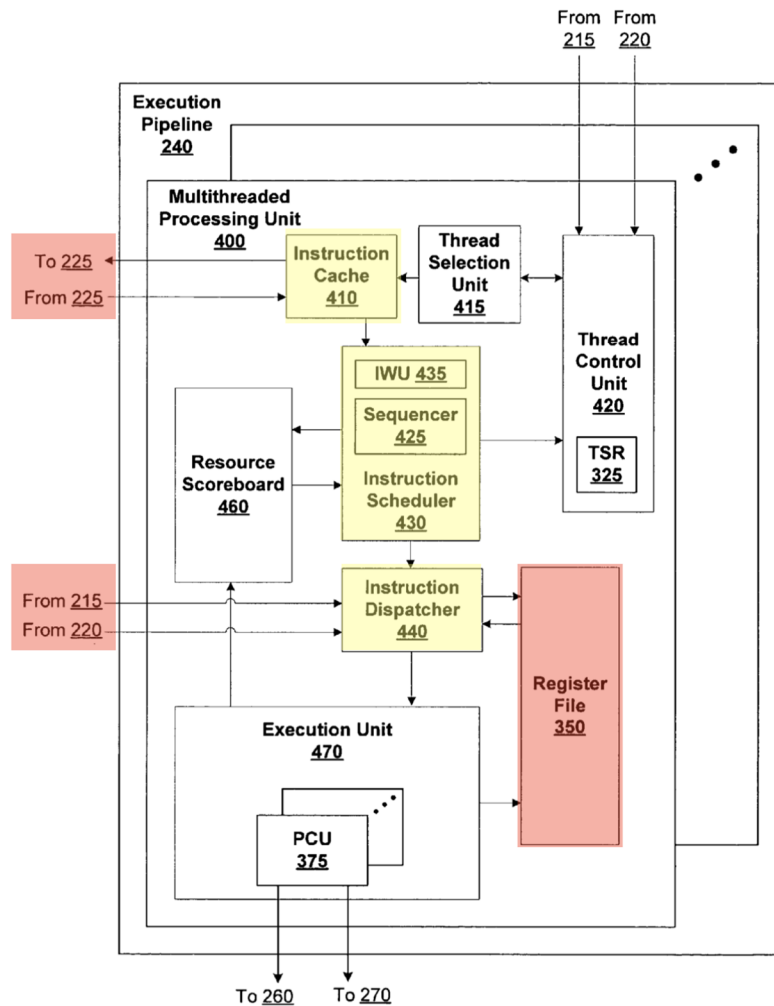


FIG. 4

55. As shown below, Fig. 2 in Lindholm ‘684 shows these external connections (highlights added). The “sequencer” in execution pipeline 240 is

connected to Pixel Input Buffer 215, Vertex Input Buffer 220, and Texture Unit 225. These buffers 215 and 220 are part of the claimed “memory” in claim 6 of the

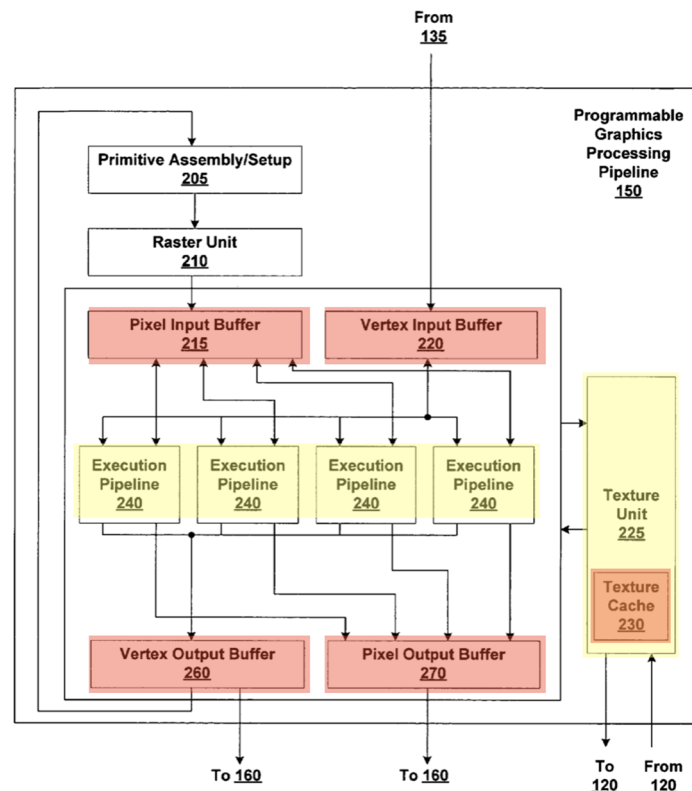


FIG. 2

‘454 patent. In addition, the sequencer is connected to the Texture Cache 230 inside the Texture Unit 225. Again, this cache is part of the claimed “memory” in claim 6. The Texture Unit 225 has connections 120 external to the Programmable Graphics Processing Pipeline 150.

56. As shown below, Fig. 1 of the ‘454 patent shows how the Programmable Graphics Processing Pipeline 150 is connected to the Memory Controller 120 (highlights added). The Memory Controller 120 is connected to

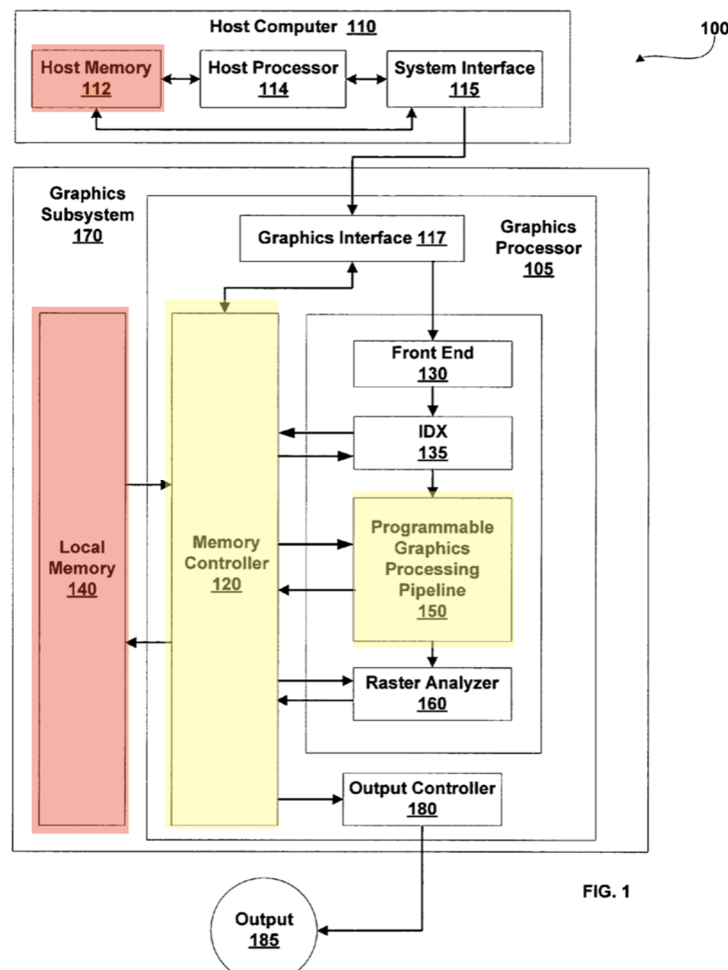


FIG. 1

Local Memory 140 and Host Memory 112, part of the “memory” in claim 6. This memory hierarchy, connected to the “sequencer,” consists of several distinct memories that can be considered “the store” or “the memory” of claims 5 and 6 in the ‘454 patent.

57. ATIs assertion that “Petitioner identifies the same components for the “memory” as for the “store” of parent claim 5” has no basis in reality. The different memories have been identified in my original declaration and supported

through quotes from the ‘454 patent description. The figures by Lindholm reproduced here clearly show several distinct memories connected to the sequencer.

**E. The Lindholm ’685 and ’913 Patents Disclose a “Control Signal” (Claims 7, 10) and “Arbiter” (Claim 10)**

58. ATI argues that Lindholm does not disclose a “control signal” of claim 7 or an “arbiter” that provides said “control signal” as required by claim 10. However, Lindholm discloses both of these, as elaborately explained in my declaration. Ex. 1003 ¶¶130-132; see also ¶¶61-63 herein.

**1. “Thread entries,” “state data,” or “priority” are control signals.**

59. ATI argues that “thread entries,” “state data,” or “priority” are not control signals. Response at p. 35. They quote the Microsoft Computer Dictionary as follows: “Control signals are “electrical signals used to control internal or external devices or processes.”” First, the Microsoft Computer Dictionary is not recognized by POSTIAs as a reference for terms of art. It is, at best, a collection of terminology that is presented in a colloquial manner for a general audience. The introduction states: “This dictionary emphasizes terminology that the average computer user will encounter...” (Ex. 2126 at Introduction).

60. Furthermore, a POSITA understands that “thread entries,” “state data,” and “priority” are represented as binary data in the memories of the processor. At its most basic, these data are binary numbers represented by the presence (“1”) or absence (“0”) of electrons as measured by voltage. These data are stored in dynamic or static memory or transmitted as signals via buses and direct connections. If one were to measure the voltage differential when these data are transferred, one would measure “electrical signals used to control internal or external devices or processes.” Therefore, at their most basic level, “thread entries,” “state data,” and “priority” are control signals. The same could be said for any other data that is stored and transmitted in digital form.

**2. Lindholm discloses the claimed “control signal” and “arbiter.”**

61. As shown in Fig. 4 of the ‘454 patent below, the ’685 patent discloses the Thread Selection Unit 415 as the claimed “selection circuit” and Thread Control Unit 420 as the claimed “arbiter.” (highlights added)

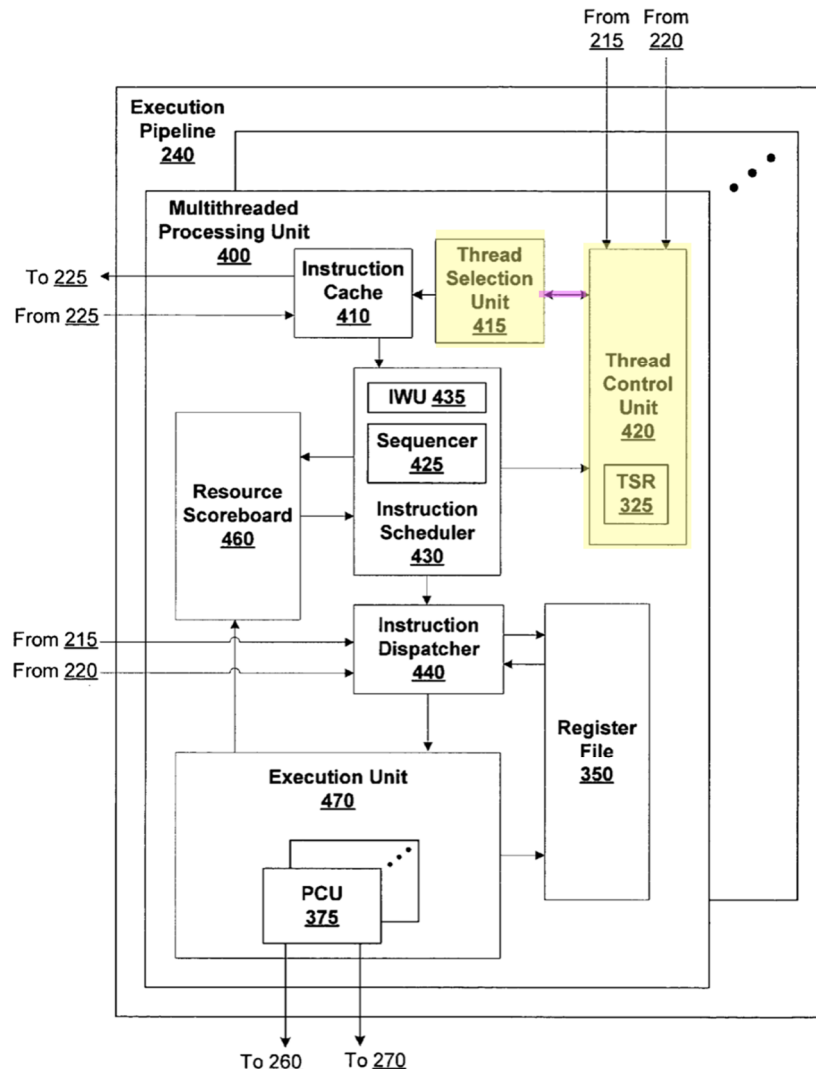


FIG. 4

62. The Thread Selection Unit functions in response to control signals (in purple) provided by the Thread Control Unit 420. Ex. 1005 at 6:60-7:5, 7:6-8:17; 15:14-17:7. These signals on the connection indicated in purple are the “electrical signals used to control internal or external devices or processes” quote by ATI. In this case, the signals from the “arbiter” (Thread Control Unit 420) control the “selection circuit” (Thread Selection Unit 415). The details of this control, and the

connection to “provide information to the store” are further detailed in my initial declaration. Ex. 1003 ¶¶130-132.

63. ATI claims: “Petitioner doesn’t identify any control signal.” Response at p. 35. However, as just explained, the control signal in Fig. 4 and its function is extensively discussed in my declaration. ATI also refers to the ‘454 patent and quotes: “[A]n appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers.” Ex. 1001 at 5:10-13. Again, this description is consistent with the signal shown in Fig. 4 and described in my declaration. ATI also claims: “Petitioner hasn’t explained how the alleged control signals are “provided by” the alleged arbiter.” Again, this is extensively described in my initial declaration. Ex. 1003 ¶¶130-132.

**3. ATI is confused by the common understanding of the term “information.”**

64. ATI also seems to be confused by the meaning of the term “information.” To quote the Microsoft Computer Dictionary: “information n. The meaning of data as it is intended to be interpreted by people. Data consists of facts, which become information when they are seen in context and convey meaning to people. Computers process data without any understanding of what the data represents.” Clearly, this definition of information does not make sense, as it

misses the obvious fact that computers do not “understand” anything. This quote corroborates my previous point (Par. 59-60) about the colloquial and non-technical nature of the dictionary.

65. Instead, a POSITA would consider the definition of information put forth by Claude Shannon, a pioneering American mathematician, electrical engineer, and cryptographer known as "the father of information theory." In his landmark paper, "A Mathematical Theory of Communication" (1948), Shannon defined information in terms of uncertainty and entropy rather than specific content or meaning. Information is quantified by how much uncertainty is reduced when a message is received. This definition is crucial in understanding how data can be efficiently transmitted across a channel because it focuses on the statistical properties of messages rather than their semantic content. In short, information is contained in digital messages, signals, and data.

66. We can apply this understanding of information to claim 7 of the ‘454 patent: “The shader of claim 5, further including a selection circuit operative to ***provide information to the store*** in response to a control signal.” (emphasis added) A POSITA would understand that information in this context means data, i.e., messages with specific statistical properties that binary numbers can encode.

67. ATI quotes from my declaration: ““Thread Selection Unit output *thread entries and other thread data, such as state data, which provides information* to the ‘store.’” Ex. 1003 ¶132. (emphasis added) The proper emphasis in this quote makes the connection between thread entries, thread data, state data, and information obvious. These data are all ‘information’ per Claude Shannon’s definition. This ‘information’ is provided to the store in response to a control signal as required by claim 7 in the ‘454 patent.

68. ATIs assertion that Lindholm does not disclose a “control signal” of claim 7 or an “arbiter” that provides said “control signal” as required by claim 10 is false.

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

#### **A. The Amantides + Kohn Patents Determine Which Data To Process By Evaluating Storage Capacity As Seen In Claims 1, 3, 4, and 5.**

69. As discussed below, it is my opinion that ATI’s arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

#### **1. The ‘454 patent does not require an “evaluation” of available storage space.**

70. I have already addressed the proper construction for the claims, which does not require any acts of “evaluation.”

**2. Amanatides + Kohn disclose a priority scheme for processing pixel and vertex data that is based on storage capacity.**

71. Even taking into account ATI's improper claim construction, the prior art renders the claims obvious.

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

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

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

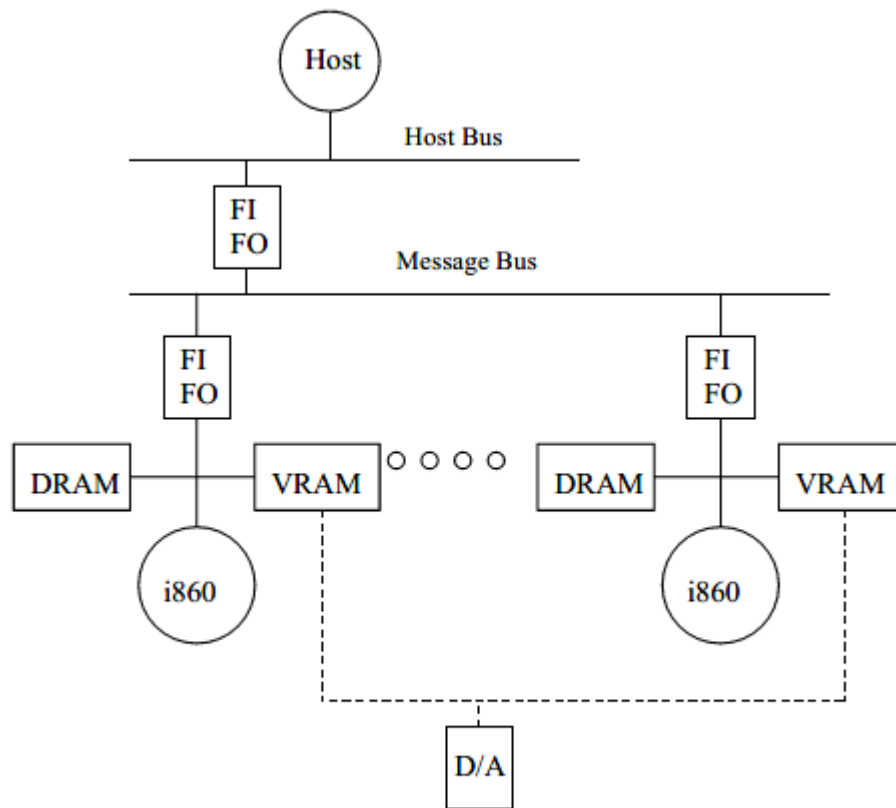


**Figure 4**

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

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

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

Ex. 1007 at 158-59.

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

74. Amanatides enabled his system to engage in automatic load balancing based on available storage space (‘454 claims 1 and 5):

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

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

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

Ex. 1007 at 158-59 (emphasis added).

**3. Amanatides + Kohn disclose processing of pixel and vertex data based on the availability of shared resources.**

75. ATI is wrong that the prior art does not disclose this limitation.

76. Amanatides was also able to accomplish auto load balancing between vertex and pixel processing in his graphics system based on shared resources ('454 claims 3 and 4):

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

...

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

...

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

Ex. 1007 at 158-59 (emphasis added).

77. The “shared resources” in Amanatides include the “general purpose register block,” (Ex. 1001 5:32-52), which in Amanatides can encompass the bus, FIFO, DRAM and VRAM for each i860 processor, as well as the instruction and data cache, memory management unit, bus control unit, core registers, floating-point registers, buses, and other circuitry within each processor, as shown in Kohn. These resources are all “shared” for both vertex and pixel processing. The bus, FIFO, DRAM and VRAM are all operatively coupled to the i860 processor, and those components together with the on-chip instruction and data cache, memory management unit, bus control unit, core registers, floating-point registers, buses, and other circuitry are operatively coupled to the functional units within the processor (which individually or collectively may also be considered the “processor unit” of the claim).

**4. The ID correctly identified the priority scheme in Amanatides + Kohn as disclosing processing of pixel and vertex data based on the availability of shared resources.**

78. As ATI points out, the ID correctly “relied on the implementation of the priority scheme in Amanatides to find it discloses the above elements. ID, 57.”

Response at p. 37

79. Amanatides was also able to accomplish auto load balancing between vertex and pixel processing in his graphics system based on shared resources (‘454 claims 3 and 4):

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

...

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

...

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

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

Ex. 1007 at 158-59 (emphasis added).

80. ATI goes to great extent to describe how GM and RM messages are sent between the processors in Amanatides. Response at p. 39-43. ATI argues that “once one GM is converted, only RMs are sent and processed.” *Id.*, at p. 39 This is false and misses the key design goals of Amanatides, which is to provide flexible load balancing between processing of GM and RM packets.

81. Amanatides’ system uses priorities and automatically load-balances its graphics processing workload for vertex and pixel data – a practice that presents various advantages: “First, the design is auto load balancing. For example, if there is a lot of texture mapping, the processors spend most of their time on this. If most of the primitives are simple, then the processors spend the majority of their time on the front end computation. Alternately, if the computation is ray tracing, no front-end processors are wasted idling. Second, the approach tends to be simpler, requiring powerful general-purpose processors instead of special-purpose graphics hardware. Because VLSI technology is making these single-chip microprocessors increasingly available this results in a smaller, simpler board layout. Finally, it is a more flexible approach. Since the hardware pipeline is not optimized for a

particular algorithm, there is a lot of freedom to change or enhance capabilities. For example, higher-order primitives can be used (NURBS). If necessary, the model can be distributed amongst the processors (assuming that they have the memory) without introducing inefficient asymmetries.” Ex. 1007 at 157.

82. Furthermore, Amanatides explicitly states that GM packets are given low priority, whereas RM packets are given high priority: “As was mentioned earlier packets on the bus have two priorities, low and high. Low priority packets cannot get onto the bus if high priority packets are waiting. If we map GMs to low priority packets and RMs to high priority packets then the solution is in sight.” Ex. 1007 at 158-59.

83. Amanatides describes that GM and RM messages can be processed out of order: “Since processors complete their tasks of converting GMs into RMs at different rates it is possible that RMs arrive at processor in the wrong priority order. The processor has to sort this out; it may have to store up to 7 RMs (this is as far off as the ordering can get).” Ex. 1007 at 159.

84. Amanatides also describes that GMs are turned into RMs in the processors: “The worst possible scenario is when every input queue is full of GMs. As soon as one of these GMs turns into a RM it shuts down further introduction of GMs from the host. The current GMs are slowly turned into RMs and wait in the

output FIFOs. As long as the output FIFOs are at least as big as the input FIFOs then each processor can do useful work. *Whenever space appears at any input FIFO it is the RMs that are delivered to it. Eventually, all the GMs will be turned into RMs and will be delivered into their appropriate input FIFOs.* Now, *when all the output FIFOs are empty and the processors are working on the remaining RMs in their input FIFOs* can the host begin to deliver GMs again. What will eventually happen is that *each processor will oscillate between working on RMs and GMs*; in both cases, it will be doing useful work. The long FIFOs (room for 32 packets) make sure that the processors are kept busy.” Ex. 1007 at 160 (emphasis added).

85. The evidence demonstrates that Amanatides not only processes GM and RM packets out of order (“oscillating”), but also makes that decision based on available storage space (“input FIFOs” or “output FIFOs”) and resources (bus, FIFOs, processors, memories). ATI is wrong on every account in their opinion. The ID is correct that Amanatides considers storage space when processing vertex and pixel data. ID, 57.

86. ATI claims: “And even if the ID were correct, the output FIFO is not part of the alleged “general purpose register” (claim 1), “shared resources” (claims 3, 4), and “store” (claim 5).” However, this misses the obvious fact that a FIFO

contains registers that can maintain both vertex and pixel data, and individually or collectively constitute “a general purpose register block” as required by claims 1 and 2 and the data “store” of claim 5. The input and output FIFOs are also “shared resources” as required by claims 3 and 4.

**5. ATI falsely imbues intelligent behavior to the graphics processing system of Amanatides.**

87. ATI argues the same point about processing of GM and RM over and over again, and then states: “But because the output FIFOs still have RMs, *the system refuses* to attempt to send, store, and process any GMs. Id. *It chooses* not to store GMs not because it cannot fit them in memory, as required by the plain claim language, but because it prefers to work on RMs until all RMs are gone to prevent deadlock.” Response at p. 41. (emphasis added)

88. This attribution of intelligent behavior such as *refusing* and *choosing* of actions to a graphics processing system is bizarre. It demonstrates ATIs desperate and failing attempts at making arguments. As the evidence clearly shows, the mechanisms described in Amanatides are not intelligent, but fulfill the requirements of claims 1, 3, 4, and 5 of the ‘454 patent.

**B. The Amantides + Kohn Patents Disclose A Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion**

**Based On Switching Between Instructions In The Instruction Store (Claim 11).**

89. As discussed below, it is my opinion that ATI's arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. Kohn discloses operations “at various degrees of completion based on switching between 30 instructions in the instruction store.”**

90. ATI seems to have a fundamental misunderstanding of how pipelining in a processor work. ATI states: “A pipeline doesn't operate perform operations at various stages of completion based on switching between instructions in the instruction store. EX. 2121, ¶348. In a pipeline, data comes in, one operation is done on it, then it moves to the next stage where another operation is done on it, etc.” Response at p. 44.

91. As explained previously, a POSITA would understand that pipelined processors, such as the i860 processor described by Kohn, use pipelining to overlap execution of *instructions*, not of *data* as ATI wrongly claims. As Kohn explains: ““The RISC core contains a pipeline consisting of four stages: fetch, decode, execute, and write.” Ex. 1008, 17. The operations fetch, decode, and execute are applied to *instructions*. Instructions are *fetched* from the instruction store; they are *decoded* into signals for the ALUs; and they are *executed* in the

ALUs (Arithmetic Logic Units) of the processor. The resulting data of the execution is the *written* back to registers. It is not the data, as ATI wrongly claims, that moves along the pipeline. It is the instructions from the instruction store that move along the pipeline at various degrees of completion.

92. Kohn notes that the floating point and integer processor units both had multiple stages, which a POSITA would have understood to mean that as different operations are selected to be loaded into the pipeline they may be at different stages of execution. Ex. 1008at 22-23. For example, Kohn describes performing operations in parallel, in a “dual-operation mode” within the “pipelined mode,” and that both floating-point operations and scalar operations could “take advantage of the pipelining capability.” *Id.* A POSITA would understand, based on the combination of Amanatides and Kohn, that the functional units within each i860 processor could not only be performing vertex operations and pixel operations simultaneously, but at various degrees of completion, based on switching between instructions in the instruction cache.

93. ATI further states: “The instructions are already loaded in the blocks that perform the instruction; there is no fetching of instructions from the instruction store, as required.” It is unclear what ATI refers to with the phrase “loaded in the blocks that perform the instruction.” One could surmise that ATI refers to the

pipeline stages. But if this were the case, then ATI concedes that instructions (not data) are being loaded and moved along the pipeline stages (“blocks”). In any case, a POSITA will know that instructions have to be fetched from the instruction store in order to decode and execute them. This is how processors work, including the i860 processor described in Kohn.

**2. A POSITA would be motivated to combine Amanatides + Kohn.**

94. ATI claims: “Moreover, there is no discussion of why using the pipeline of Kohn would be in any way relevant or beneficial to Amanatides, including the relied-upon load balancing and deadlock solution, or any other part of Amanatides, or even how the proposed combination would work.” Response at p. 44. This is false, and ATI deliberately ignores statements made in my declaration.

95. As described in my initial declaration (Ex. 1003 ¶¶165-166), a POSITA would be motivated to combine Amanatides with Kohn because both references use the Intel i860 processor and both references describe processing graphics with that same processor. The Kohn reference simply provides details regarding the same Intel i860 processor that was used in Amanatides. One of ordinary skill in the art would have been specifically motivated to combine the two references because they are directed to the same processor.

96. ATI is clutching at straws and further states: “Even if some concoction could be made, Petitioner hasn’t adequately explained it or why it would be obvious and motivated.” Response at p. 44. Since Amanatides explicitly uses the i860 processor in his system, there is no need for a “concoction.” There is also no need to explain to a POSITA why combining Amanatides + Kohn is obvious since both describe graphics processing with the same i860 processor.

**C. Amantides + Kohn Disclose “Selected Data” (claims 2 and 5)**

97. Claims 2 and 5 recite “selected data” that is from or stored/maintained in a general purpose register block or store. Nothing in the claims identifies how, when, or by what component or process that data is selected. Thus, a POSITA, reading the ‘454 patent and claims, would not understand there to be any “selection” action required by the claim as asserted by ATI, only that “selected data” exist and be utilized. Likewise, a POSITA would not understand there to be any requirement in the claims for “transmitting” data as asserted by ATI. A POSITA would understand that the ‘454 patent is silent on how the processing unit obtains the selected data from the general purpose register block or store, and, thus, any mechanism is possible. That is, the processing unit may simply have access to the data sitting in the general purpose register block, or it

may retrieve the data itself, or it may receive the data. No one way is required or even preferred.

98. ATI states: “But the reliance by Petitioner and the ID on the “priority” of certain data over others in Amanatides doesn’t establish a means by which data is selected “from the general purpose register block” (claim 2) or from “a store” (claim 5). EX. 2121, ¶351.” ATI is wrong.

**1. Amanatides does not impose any temporal restrictions of the processing of GM and RM packets.**

99. ATI states: “There is no selection of GMs versus RMs sitting in any alleged register block or data store. EX. 2121, ¶352. Rather, as explained in the previous section, there are different modes of operation: at first GM only, then RM only, then GM only. Id. The system then “oscillate[s]” back and forth between these modes. EX. 1007, 160.” Response at p. 45. However, Amanatides does *not* require that processing of GM and RM packets cannot overlap. For example, it is possible that the input FIFOs still contain RM packets while the processor is starting to process GM packets, and vice versa.

100. Amanatides writes: “The worst possible scenario is when every input queue is full of GMs. As soon as one of these GMs turns into a RM it shuts down further introduction of GMs from the host. The current GMs are slowly turned into RMs and wait in the output FIFOs. As long as the output FIFOs are at least as big

as the input FIFOs then each processor can do useful work. ***Whenever space appears at any input FIFO it is the RMs that are delivered to it.*** Eventually, all the GMs will be turned into RMs and will be delivered into their appropriate input FIFOs. Now, ***when all the output FIFOs are empty and the processors are working on the remaining RMs in their input FIFOs*** can the host begin to deliver GMs again. What will eventually happen is that each processor will oscillate between working on RMs and GMs; in both cases, it will be doing useful work. The long FIFOs (room for 32 packets) make sure that the processors are kept busy.” Ex. 1007 at 160 (emphasis added).

101. In other words, each of Amanatides unified shader processors check the input FIFO for available work. It processes whatever packet is available (GM or RM) and puts the resulting packet into the output FIFO. It is entirely possible that a mix of GM and RM packets are stored in the input or output FIFOs during operation. It is further possible that different processors process the data at different rates, e.g., taking longer for the processing of a GM packet than other processors. Processors can send packets to other processors at any time: “Since every processor can write into every other processor’s input FIFO at the same time there is a remote possibility of deadlock.” Ex. 1007 at 159.

102. ATI is wrong in claiming that “there are different modes of operation: at first GM only, then RM only, then GM only.” ATI does not specify what exactly is meant by the term “only,” but it seems to indicate some temporal restriction. ATI’s use of the phrase “GM-only phases” and “RM-only phases” is improper and goes counter to the invention and disclosure of Amanatides. Amanatides explicitly aims at a “flexible” graphics architecture that can process GM and RM packets simultaneously at different stages of completion.

**2. Reading data from FIFOs in Amanatides is “selecting data” from a data store.**

103. ATI states: “Thus, there is no world in which GMs and RMs are sitting in a register block or data store and are selected from, which is the apparent read by Petitioner and the ID. EX. 2121, ¶352.”

104. As described in my initial declaration, undisputed by ATI, there are several elements in Amanatides that disclose register blocks and data stores. Amanatides discloses a DRAM that is operatively coupled to the processor unit. Ex. 1007 at 158. Amanatides also discloses VRAM associated with each i860 processor. *Id.*, at 157-58. Amanatides further discloses a bus and an input FIFO for each processor. *Id.*, at 158-59. Each i860 processor also has an on-chip cache. *Id.* See also Ex. 1008 at 15. Accordingly, each of these components contain registers

that individually or collectively constitute “a general purpose register block” as required by claims 1 and 2 and the data “store” of claim 5.

105. As mentioned before, Amanatides explicitly discloses that GM and RM packets are stored in the output and input FIFOs. These FIFOs contain registers that can maintain both GM packets (e.g., vertex data) and RM packets (e.g., pixel data). Using ATIs words, these GM and RM packets are “sitting” in the FIFO registers or data stores and can be “selected from” by the processors. The mechanisms by which data is “selected from” a FIFO is well known to a POSITA.

106. During graphics processing, the GM and RM packets, and their associated data (e.g., vertex and pixel data), may at some point also be stored in any of the other “general purpose register blocks” or “data stores” in Amanatides. For example, it is inevitable that data would be loaded into the on-chip cache and internal registers of the i860 processor. Once it is stored (or “sitting”) there, it would be “selected from” by the internal circuitry and ALUs in the i860 processor. All of this is well known to a POSITA.

107. Similarly, graphics data (messages, vertex, and pixel data) may also be stored in the DRAM and VRAM memories that are connected to each processor. One reason for this is, as Amanatides notes: “The new design would require more memory for each processor; a lot of memory is needed as textures

and geometric models, if they are stored locally, are space intensive. ... DRAM also has the advantage that a lot more memory can be placed per unit area.” Ex. 1007 at 158. In other words, the DRAM in Amanatides is used to store textures (pixel data) and geometric models (vertex data), among other things. Similarly, the VRAM in Amanatides is used for a distributed frame buffer to store pixel data: “A similar number of VRAM chips would be required for the distributed frame buffer.” Ex. 1007 at 158. A POSITA would understand that either of these memories (the “store”) can store vertex and pixel data, including GMs and RMs, and that the processor can “select” data from these memory locations by addressing the memories.

**3. Claims 7 and 10 do not require that the “Selection Circuit” is within the processor of the “Unified Shader”.**

108. ATI alleges that “the alleged unified shader doesn’t include the alleged selection circuit, and thus the ground fails for claims 7 and 10. EX. 2121, ¶353.” Response at p. 46. ATI then argues: “Petitioner alleges the unified shader is each G/R processor, which is equivalent to each i860 processor,...” and: “Petitioner then alleges that the selection circuit is “met by at least the bus and FIFOs, which are operative to provide packets (‘information’) to the DRAM, VRAM, and on-chip data cache (‘store’).” Pet., 59. But the bus and FIFOs are not part of the alleged unified shader. EX. 2121, ¶355.”

**4. Amanatides and Kohn include the “Selection Circuit” within the “Unified Shader” as required by claims 7 and 10.**

109. Even if the preamble is limiting, ATI is not interpreting the claims correctly. Claim 5 in the ‘454 patent reads:

“A unified shader comprising:

a processor unit;

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

110. Claim 5 clearly states that the unified shader includes a processor unit and a sequencer. ATI does not contest that Amanatides and Kohn meet this limitation. The unified shader in Amanatides is the G/R unit, which is the i860 processor of Kohn. As explained in my initial declaration: “The on-chip cache, together with the Memory Management Unit (Ex. 1008 at 15, 17, 20), comprise the claimed “sequencer” and sequencer “circuitry” of the ‘454 patent.” Ex. 1003 ¶¶178-179. So the sequencer and processor unit in Amanatides and Kohn comprise the unified shader.

111. Claim 7 in the ‘454 patent reads:

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

112. As explained in my initial declaration, the selection circuit in Amanatides and Kohn is the priority allocation system, which involves the i860 processors. “As explained supra at claim 3[a] and [c], the combination of Amanatides and Kohn includes an arbiter that determines which input (e.g., vertex or pixel) to provide to the processor units. Amanatides utilizes a priority allocation system in which priority is given to one of the types of graphics data (e.g., pixels) so that pixel samples are processed until none remain or until the processor cannot store additional pixel samples. Some packets (vertex or pixel) can be assigned a high priority, and other packets assigned a low priority. “Low priority packets cannot get onto the bus if high priority packets are waiting.” Ex. 1007 at 158-59.” And: “Assignment of low-priority and high-priority packets to the time slots of the bus is described by Amanatides as “arbitration.” Ex. 1007 at 159.” Ex. 1003 ¶¶195-196.

113. It is understood by a POSITA that this “arbitration” of packets onto the message passing bus is performed by the processors and involves the bus and FIFOs: “Accordingly, the “selection circuit” of claim 7 is met by *at least* the bus and FIFOs, which are operative to provide packets (“information”) to the DRAM,

VRAM, and ***on-chip data cache*** (“store”) in response to a control signal, such as whether the packets are designated high-priority or low priority as explained further below.” *Id.* In other words, the “selection circuit” of Amanatides is included in the unified shader (the processor) by virtue of the priority scheme and the on-chip data cache.

**D. Amantides + Kohn Disclose a “Control Signal” (claims 7 and 10)**

114. As discussed below, it is my opinion that ATI’s arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. Packets in Amanatides and Kohn include a priority.**

115. ATI claims that: “First, there is no disclosure that the packets themselves contain a priority designation. EX. 2121, ¶359. Petitioner’s theory is that GMs are given low priority and RMs are given high priority. A POSITA would understand that given this preset priority scheme, it would be a waste of resources to list the priority in the packet. *Id.* Rather, the system would simply treat the different packets differently based on type. *Id.*” Response at p. 49.

116. It is clear, however, that Amantides states that priority is part of message packets: “The message bus has the capability of both high and low priority packets (the need for which we will describe later).” The terms “high

priority packets” and “low priority packets” have clear meaning, and a POSITA would understand the different priorities have to be part of the packet.

117. Furthermore, nothing in Amanatides or Kohn suggests that these priorities are a “preset priority scheme,” as ATI claims. Amanatides discloses a flexible graphics system, and making priorities “preset” is the opposite of a flexible design. Even if ATI is correct, the packets would at a minimum have to include a priority bit, e.g., “0” for low and “1” for high priority packets. This priority bit, or the priority bits, are the control signal required by claims 7 and 10.

118. It is not clear what ATI means by packet “type,” as this term never appears in any of the patents or Amanatides and Kohn. However, it is possible that ATI means that packet types could be “geometry messages” (GM) or “rendering messages” (RM). To indicate these packet types would require at least a “packet type” bit, e.g., “0” for GM and “1” for RM. If this were the case, which is false, then these type bits would be the control signal required by claims 7 and 10. The reason this is false is that Amanatides clearly distinguishes between packet “types” (GM and RM) and “priority.” Furthermore, Amanatides only uses “type” of packets in the following sense: “There are two *types* of packets: high and low priority packets.” Ex. 1007 at 159 (emphasis added).

**2. ATI is relies on colloquial sources and non-technical terminology.**

119. ATI again refers to the unreliable and colloquial Microsoft Computer Dictionary and their “definition” of control signal. “In contrast, a POSITA would understand control signals are “electrical signals used to control internal or external devices or processes.” EX. 2126, 128; EX. 2121, ¶360.” Response at p. 50. As explained earlier this “definition” is consistent with the disclosure of control signal in Amanatides, where priorities are control signals.

120. Furthermore, ATI quotes the dictionary: “A control signal is an instruction from the sender to the recipient, not mere data that the recipient can interpret however it chooses. EX. 2121, ¶360.” This “definition” does not make sense. A POSITA knows that a “control signal” is not the same as an “instruction,” at least in the technical sense. It appears that the Microsoft Computer Dictionary uses the term “instruction” here in the colloquial sense, as in a sender sending some “instruction” (e.g., ‘open the door’) to a recipient. This is not a technical explanation, and it has no place in an IPR case.

121. Furthermore, ATI seems to agree with Microsoft that “an instruction” is not “mere data that the recipient can interpret however it chooses.” Again, one has to assume that “instruction” here is not used in the technical (e.g., as a processor instruction) but in the colloquial sense. In either case, ATI does not

explain what “interpret however it chooses” means. One must assume that “interpret” is used in the colloquial sense, meaning some form of human interpretation. Once again, ATI fails to make a technical argument and instead clings to fuzzy concepts from an unreliable source – at least in terms of the level of technical expertise a POSITA would require for the subject matter of the present patent and the level of ordinary skill in the art required.

**E. Amantides + Kohn Disclose a “Selection Circuit” and “Arbiter,” (claim 10)**

122. As discussed below, it is my opinion that ATI’s arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. ATI fails to understand the basic circuits of FIFOs and buses.**

123. ATI claims: “But neither the bus nor the FIFOs can be “selection circuits.” EX. 2121, ¶362 FIFOs are merely memory, not selection circuits. Id. And bus lines are merely transmission lines, not selection circuits. Id.” Response at p. 50-51. ATI fails to understand the basic designs of FIFOs and communication buses in processor systems.

124. A POSITA also knows that FIFOs are not just “merely memory” as ATI claims. The circuit components of a FIFO typically include:

*Memory or Registers:* This is the storage area where the data is held. It can be constructed using registers or dual-port RAM for larger FIFOs.

*Read Pointer and Write Pointer:* These pointers keep track of the locations where data should next be read from or written to. The write pointer tracks the next available memory location for incoming data, and the read pointer tracks the location of the oldest data not yet read.

*Control Logic:* This includes the circuitry that manages the pointers and determines when the FIFO is full or empty. It also controls the read and write operations based on the status signals.

*Status Flags:* Common flags include the Full Flag (FIFO has no more space to store new data) and Empty Flag (FIFO contains no data to read).

125. Similarly, a POSITA would know that a communication bus is not “merely transmission lines” as ATI claims. Buses operate based on specific protocols, which define the rules and timings for data transfers. These protocols manage how devices on the bus request access to the data lines, how conflicts are resolved, and how data integrity is maintained during transmission. When multiple devices need to use the bus at the same time, a method of bus arbitration is necessary to prevent conflicts. This can be managed by a dedicated bus controller

or through a distributed arbitration method. Thus, in my opinion, a POSITA would understand that Amanatides' bus and FIFOs render the claimed selection circuit obvious.

**2. Packets in Amanatides are sent from FIFOs over the bus to DRAM, VRAM, and the on-chip cache.**

126. ATI states: "There is no indication in Amanatides that "packets" are ever sent from the FIFOs or bus to DRAM, VRAM, or on-chip cache." Response at p. 51. This is false.

127. Amanatides describes how packets are sent from FIFOs onto the bus: "Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. As well, there is a fair scheduling policy in which *a second packet from any FIFO cannot get onto the bus* until all other *FIFOs are first given a chance to send theirs.*" Ex. 1007 at 159. (emphasis added)

128. Amanatides describes that packets are received and processed by the processors and stored in VRAM: "Each processor waits for packets in its input FIFO. Packets containing polygons come in two flavors: geometry messages (GM) and rendering messages (RM). The GMs come from the host and their polygons are transformed, clipped, shaded and then sent out as RMs via the output FIFO *to the other processors.* When *a processor receives a RM* it *scan converts* the polygon and *draws it in its frame buffer.*" Ex. 1007 at 159 (emphasis added). In

this case, a RM packet is received by a processor and then processed via scan conversion of a polygon and drawing the resulting fragments or pixels in the frame buffer.

129. A POSITA will understand that receiving and processing packets in the i860 processor of Amanatides and Kohn involves storing the packet or at least the data within the packet in the on-chip cache. Furthermore, scan converting and storing data in the frame buffer by the processor involves storing packets or at least data contained within packets in VRAM because the frame buffer in Amanatides is comprised of VRAM: “A similar number of VRAM chips would be required for the distributed frame buffer.” Ex. 1007 at 158.

130. Amanatides also discloses that DRAM is used to store graphics data: “The new design would require more memory for each processor; a lot of memory is needed as *textures and geometric models*, if they are stored locally, are space intensive. As well, program space needs to be larger as each processor does more work. To keep costs down, we wanted to eliminate the need for SRAM, and *use DRAM instead*.” Ex. 1007 at 158. (emphasis added) Since graphics data in Amanatides is sent via messages (GM and RM), a POSITA would understand that these messages, or at least the data contained within the messages, can be stored in

DRAM, which is the “local memory” of the processors in Amanatides. Ex. 1007 at 160.

**3. The “control signal” in Amanatides prompts information to be transmitted and stored.**

131. ATI claims: “Moreover, there is no indication or analysis proving that the alleged “control signal” (which is alleged to be a part of the packet itself) prompts such a transmission, as required. Id.” Response at p. 51.

132. Claim 7 in the ‘454 patent reads: “The shader of claim 5, further including a selection circuit operative to provide information to the store in response to a control signal.” Nothing in the claim requires that the control signal “prompts such a transmission” as ATI seems to suggest. The claim only requires that information to the store is provided “in response to a control signal.” The claim does not specify when or how the information is provided to the store.

133. As explained earlier herein and in my initial declaration, the “control signal” in Amanatides and Kohn is part of the packet, e.g., as “low” or “high” priority. Amanatides describes the packets that are transferred on the message passing bus between processors. “There are two types of packets: high and low priority packets. Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs. As well, there is a fair

scheduling policy in which a second packet from any FIFO cannot get onto the bus until all other FIFOs are first given a chance to send theirs.” Ex. 1007 at 159.

134. Amanatides makes it clear that the priority is the “control signal” that indicates when packets are sent from FIFOs, e.g., “Low priority packets get onto the bus only if no high-priority packets are waiting to be sent in any of the other FIFOs”. These packets “get onto the bus” and are stored in FIFOs: “There is room for 32 packets in each FIFO. This allows for considerable incoming work.” Ex. 1007 at 159. In other words, the “control signal” (priority) in Amanatides prompts information (packets) to be provided to the store (FIFO and other memory). At a minimum, a POSITA would understand Amanatides’ disclosure to render obvious that the priority information is part of the packets even if not expressly stated.

**4. Claim 10 does not require specific timing for arbitration.**

135. In their response, ATI makes incoherent arguments about the timing of selection and arbitration. ATI seems to agree with Petitioner that Amanatides discloses an arbiter. ATI writes: “But in this system, the alleged *arbitration is separate from and happens after selection*; the decision is made and then the final sorting onto the bus via timeslots happens by action of the undisclosed arbiter, as a POSITA would know. EX. 2123, 82:6-83:8; EX. 2121, ¶365.” Response p. 51-52. (emphasis added)

136. Claim 10 in the '454 patent states: "The shader of claim 7, wherein the control signal is provided by an arbiter." There is nothing in the claim that specifies the timing of arbitration. That is, the claim does not specify when arbitration happens with respect to selection. The arbitration could happen at the same time, before, or after selection. The fact that ATI seems to argue that arbitration happens "after selection" is irrelevant to meeting the limitation of claim 10.

137. ATI then states: "At a minimum, that is the most obvious way to implement the arbiter, and without question there can be no inherency argument that it must be done in the opposite order, as would be required for Petitioner's theory." Again, because timing is not a requirement, ATI's argument is irrelevant. All that claim 10 requires is that the control signal come from an arbiter. There is no requirement as to what constitutes an arbiter or what an arbiter must do other than send a control signal. A POSITA would understand that any structure or software or combination thereof that does arbitration is an arbiter. The Petition identifies that Amanatides has a selection circuit that performs some arbitration (which ATI admits), e.g., decision of what type of data to transmit, which is all that is required under claim 10. Pet. 59-60.

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

**A. The Selzer + Fiske Patents Determine Which Data To Process By Evaluating Storage Capacity As Seen In Claims 1, 3, 4, and 5.**

138. As discussed below, it is my opinion that ATI's arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. The '454 patent does not require an "evaluation" of available storage space.**

139. I have already addressed the improper reading of the claim by ATI.

**2. Selzer discloses processing of pixel and vertex data based on storage capacity.**

140. The only basis on which the Board found the combination of Selzer and Fiske lacking for claims 1 and 3-11 was that the Petition failed to demonstrate how Selzer's "load balancing teachings" disclosed continuing pixel processing until shared resources become available, and vice versa, or how Fiske's "switching between unfinished threads" and thread prioritization demonstrates switching based on the amount of space available in shared resources. ID 71-72. First, as explained Section II.D, the claim does not require that the *unified shader* must (1) "consider" or "evaluate" available space, and (2) "determine" execution based on that "consideration." POR 36-43. *See* Section II.D (functions not required by claims, and moreover, performed by arbiter outside unified shader). Second, Selzer discloses

“load balancing” that processes pixel and vertex data based on availability of shared resources or storage. “The capability of automatically distributing the workload between geometry and rendering modules means inherently task switching between two jobs within the same application. Supported by the large local memory ... the switching is reduced to *the saving and restoring of all processor registers*, processing interrupt control and *inspecting the rendering FIFO status*.” EX. 1016, 47-48 (emphasis added). “After rendering an object any processor may run into an idle state *if there is no rendering data in his input buffer*.” *Id.*, 47 (emphasis added). In Selzer, the rendering module is a unified shader, and its FIFO (input buffer) is a “shared resource” (claims 3 and 4), “store” (claim 5) and part of the “general purpose register block” (claim 1) because it can store either vertex or pixel tasks. EX. 1003, ¶¶218-220. The Board recognized (in considering *Amanatides*) that a FIFO can be the general register (claim 1), shared resources (claims 3-4), or store (claims 5-10). ID 57. A “balancing mechanism [can be] activated” in Selzer at any point in time to achieve load balancing, *e.g.*, activating geometry processing in the rendering modules. EX. 1016, 48. If the RM “*inspects*” the FIFO and there are no pixel tasks, then that shared resource/register/store is available for vertex tasks and it switches over. [cite, Pfister] Selzer switches freely between processing vertices and pixels, and in each case, first “inspects” the FIFO.

*Id.* Selzer allows this flexible processing of geometry (G) and pixel (P) jobs, such that the processing might proceed in any sequence. *Id.* It is therefore obvious to a POSITA that the flexible load balancing of the RM will switch back and forth between rendering/pixel tasks and geometry/vertex tasks based on the availability of shared resources, store, or general register. Selzer further discloses that its master module may direct vertex or pixel data to a rendering module in order to load balance, *i.e.*, determine what to process based consideration of available space. A POSITA would also understand that “the saving and restoring of all processing registers” in Selzer would include inspection of available space and shared resources. A POSITA understands that when Selzer switches between geometry/vertex and rendering/pixel tasks, the RM must load the new geometry or pixel data for the new task into the registers, and thus must consider the availability and content of the registers when switching. A POSITA would also understand that Fiske discloses processing pixel and vertex data based on storage capacity (shared resources) in connection with its hardware context switching. Pet. 63; EX. 1010, 211. As I have explained, each thread in Fiske must have its own hardware context, *e.g.*, constant values and state data. When switching between vertex and pixel threads, the processor must save its current context and data to memory and load another context and data from memory. A POSITA would have understood these

switches depend on considering the availability of storage capacity. EX. 1010, 211. Claim 11 is distinct from other claims because it only requires performing pixel/vertex operations “based on vertex or pixel workload,” not the availability of memory or shared resources. As explained above, Selzer and Fiske clearly disclose switching between pixel or vertex operations based on workload. ATI also argues that claim 1 requires ceasing vertex operations and then continuing pixel operations. POR 55. This is wrong, *supra* Section III.B., and contradicts the ’454 patent’s alleged improvement of flexible operations over the prior art. EX. 1001, 2:25-30.

141. Further, ATI admits: “Selzer explains it prefers that its rendering module work on rendering calculations, and switches only to geometry calculations when the rendering module’s input buffer is empty, as this approach operates at “peak performance.” EX. 1016, 47-48.” Response at p. 53. Selzer writes: “After rendering an object any processor may run into an idle state ***if there is no rendering data in his input buffer***. Performing a task switch it will ***request new unprocessed geometry objects and continue geometry calculations***.” Ex. 1016, 47. (emphasis added)

142. As explained in my initial declaration, the input buffers (e.g., programmable memories and FIFOs) in Selzer is part of the “general purpose register block” of claim 1, the “shared resources” of claims 3 and 4, and the data

“store” of claim 5 in the ‘454 patent. Ex. 1003 ¶¶258, 229-230. Therefore, the section above in Selzer, which is also quoted by ATI, discloses evaluation of “available storage space in determining what to process.” Response at p. 53.

**3. Fiske’s hardware context switch discloses processing of pixel and vertex data based on storage capacity.**

143. ATI argues: “There is no assessment of available space for storage that impacts Fiske’s priority scheme as required by the claims. EX. 2121, ¶369.” Response at p. 53. ATI is wrong.

144. As explained in my initial declaration, and as admitted by ATI in their response, Fiske discloses a thread priority scheme: “Thread prioritization involves assigning a priority to all the different *threads* in an application, and then using this priority to make thread scheduling decisions. The priority reflects the importance of a single thread to the completion of a single application. ... [I]n our case the priority is used to make scheduling decisions on every *hardware context switch* in a multiple-context processor.” Fiske p. 211. (emphasis added)

145. A POSITA would understand the meaning of the terms “threads” and “hardware context switch.” The multithreaded processor in Fiske executes multiple threads of instructions to improve performance. A *thread* is a sequential set of instructions that specify what operations to perform, such as adding two values, accessing memory, comparing values with each other, etc. A graphics program in

Fiske consists of multiple threads, for example, one for processing vertex data and one for processing pixel data. Each thread must have its own *hardware context*, such as constant values and state data stored in the processor's registers as well as pointers to graphics data in memory that are relevant for processing the thread. When switching threads (e.g., switching between processing of vertex and pixel threads), the processor must save its current state and data to memory for the hardware context and load another state and data from memory into the hardware context. This is what Fiske calls a *hardware context switch*.

146. Fiske discloses several scenarios where such a hardware context switch may occur. Importantly, all these scenarios depend on the availability of storage capacity in the processor. For example: "If a thread is waiting for a memory reference to be satisfied then it is stalled and is not considered for scheduling until the memory reference is satisfied." Fiske p. 211. Or: "A context switch can occur on a cache miss, on a failed synchronization test, or on a change of priority of one of the threads on the processor." Fiske p. 211. Fiske explains that these hardware context switches are decided in each processor. "Note as well that thread scheduling as defined here is purely a local operation. Each processor has its own set of threads, and schedules only these." Fiske p. 211. These thread context

switches based on storage capacity are familiar to a POSITA since they occur in any multithreaded processors.

147. As the evidence shows, and as a POSITA would know, Fiske discloses processing of pixel and vertex data based on storage capacity. Furthermore, the combination of Selzer + Fiske discloses the claim limitation of the “general purpose register block” of claim 1, the “shared resources” of claims 3 and 4, and the data “store” of claim 5 in the ‘454 patent. All of the discussion by ATI about load balancing between the rendering modules is irrelevant to this issue.

**4. Claim 1 in the ‘454 patent does not impose timing constraints or ordering on the processing of vertex and pixel data.**

148. ATI states: “Claim 1 requires performing vertex operations until there is no more room in the general purpose register block for more incoming vertex data, in which case continuing to perform pixel operations until enough space appears to once again store vertex data. Id.” This is false.

149. Claim 1 states:

“A method carried out by a unified shader comprising:

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

unified shader unless the general purpose register block does not have enough available space therein to store incoming vertex data; and continuing pixel calculation operations that are to be or are currently being performed by the processor based on instructions maintained in an instruction store until enough registers within the general purpose register block become available.”

150. The first limitation in the claim is about “performing vertex manipulation operations and pixel manipulation operations.” (emphasis added) This limitation does not imply any constraints on the timing of the vertex and pixel processing. The vertex manipulation operations could be performed at the same time, before, or after the pixel manipulation operations.

151. The second limitation in the claim states “continuing pixel calculation operations that are to be or are currently being performed by the processor” without any reference to the vertex operations of the first limitation. It simply requires “continuing” the pixel calculation operations that will be performed (“are to be”) or that are “currently being performed.” It does not require to perform these pixel operations after the vertex operations, as ATI is suggesting.

152. ATI writes: “There is no disclosure at all when to *cease sending, storing, or operating on the alleged vertex data*, much less any description of

doing so based on not having room for vertex data in the general purpose register block, nor any description of *continuing to execute pixel processing in that scenario*.” Response at p. 55 (emphasis added) Again, ATI is wrongly imposing a timing constraint on the processing of vertex and pixel data that is not part of claim 1 in the ‘454 patent. The claim does **not** require that pixel processing needs to continue after vertex processing ceases.

153. ATI further claims: “There is also no description of determining a second time at which there is again room to store vertex data. Id.” Response at p. 55. It is not clear what ATI means by “a second time.” There is nothing in claim 1 of the ‘454 patent that requires a specific timing.

154. The inventors of the ‘454 patent did not consider any timing constraints. On the contrary, their alleged invention was made to overcome timing constraints in the prior art. The term “time” or “timing” do not appear in connecting with processing of vertex and pixel data in the ‘454 patent. The patent inventors explicitly distinguish their alleged invention from the prior art, which seems to impose such a timing constraint: “In conventional graphics processors, the vertex shader and the pixel shader are juxtaposed in a sequential, pipelined fashion, with the vertex shader being positioned before and *operating on vertex data before the pixel shader can operate on individual pixel data*.” Ex. 1001,

2:25-30. The alleged invention of the unified shader removes this timing constraint between vertex and pixel processing, which is not present in any of the claims.

**5. Selzer + Fiske do not impose any timing constraints on the processing of vertex and pixel data.**

155. ATI wrongly claims that Selzer imposes timing constraints on the processing of vertex and pixel data. “Rather, in Selzer, the rendering module (the alleged unified shader) will only begin to process geometry operations if it has no rendering operations to perform. EX. 1016, 47. That is, like other load balancing schemes, the only consideration of when to switch is how much work of a given type there is to do. EX. 2121, ¶371. There is no disclosure in Selzer that it switches from rendering operations to geometry operations on the basis of having enough available space for geometry operations in any of Petitioner’s identified memories.” Response at p. 54.

156. As explained in the prior section (Sec. IV.A.4.), there is nothing in the ‘454 patent that imposes a timing constraint on the processing of pixel and vertex data. That is, it is improper by ATI to mention “switches from rendering operations to geometry operations on the basis of having enough available space for geometry operations.” A POSITA would not understand the claims in the ‘454 patent to require this, and the patent inventors explicitly state that their alleged invention of the unified shader overcomes this limitation of the prior art.

157. As discussed earlier, Selzer discloses ways in which the processor can switch between vertex and pixel processing tasks for dynamic load balancing based on storage capacity. For example: “After rendering an object any processor may run into an idle state if there is no rendering data in his input buffer. Performing a task switch it will request new unprocessed geometry objects and continue geometry calculations.” Ex. 1016 at p. 47. Similarly, as discussed herein, Fiske also discloses ways to switch between vertex and pixel thread processing based on available storage capacity.

158. Nothing in Selzer prevents the processor from processing vertex and pixel data in any order, i.e., the processing of vertex data could happen before, at the same time, or after the processing of pixel data. Selzer calls this dynamic load balancing, and it is the focus and goal of the article. “To enable such a dynamic load balancing and to speed up geometry calculation dynamically if required, the rendering modules are capable of performing all the geometry calculations, too.” Ex. 1016 at p. 46-47. And: “After rendering an object any processor may run into an idle state if there is no rendering data in his input buffer. Performing a task switch it will request new unprocessed geometry objects and continue geometry calculations. In this way an automatical load balancing is achieved across all the processors.” Ex. 1016 at p. 47.

159. Similarly, nothing in Fiske prevents the processor from processing vertex and pixel threads in any order. For example: “When several threads have the same priority, the hardware scheduler chooses them in round robin fashion.” Ex. 1010 at p. 212.

**6. Selzer + Fiske discloses performing pixel calculation operations and then vertex calculation operations based on resources and vice versa (claims 3 + 4).**

160. ATI states: “Claim 3 requires the processor unit to be “operative to perform pixel calculation operations until enough shared resources become available and then use the shared resources to perform vertex calculation operations.” EX. 2121, ¶373.” Response at p. 55. ATI left out the beginning of the claim limitation: “the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform...”

161. Similarly, ATI states: “Claim 4 is like claim 3 but in the alternative direction; it requires the processor unit to be “operative to perform vertex calculation operations until enough shared resources become available and then use the shared resources to perform pixel calculation operations.” EX. 2121, ¶374.” Response at p. 56. ATI left out the beginning of the claim limitation: “the processor unit operative to use the shared resources for either vertex data or pixel information and operative to perform...”

162. Similar to Claim 1, ATI attempts to argue that Selzer + Fiske impose timing restrictions on the processing of vertex and pixel data: “Similar to the above, the triggering event in Selzer in going from the alleged pixel calculations (rendering data) to the alleged vertex calculations (geometry data),...” Response at p. 55. And: “There is no disclosure in Selzer of processing the alleged vertex data (geometry data) until there is enough room for processing the alleged pixel data (rendering data).” Response at p. 56. As discussed earlier, Selzer + Fiske does not impose any timing restrictions on the processing of vertex and pixel data. Either data can be processed in any order by the unified shader in Selzer + Fiske, thereby meeting the limitation of claim 3 and 4.

163. ATI further tries to argue that the processing in Selzer + Fiske has: “...nothing to do with a determination that at a given time, there is finally enough shared resources for geometry data; the only consideration is that there is no more rendering data to operate on.” First, neither claim 3 or 4 require “enough shared resources for geometry data.” However, ATI left out parts of the claim language, which states: “the shared resources *for either vertex data or pixel information.*” (emphasis added) Clearly, the claims do not require that there are shared resources for just geometry data or just pixel data.

164. ATI also tries to argue: “There may have been sufficient resources before that happens, but the system has no way of knowing, and thus no way of “perform[ing] pixel calculation operations *until* enough shared resources become available and then us[ing] the shared resources to perform vertex calculation operations.” Response at p. 56. (emphasis and edits by ATI) This is incorrect.

165. As explained in Sections IV.A.2.-3., Selzer + Fiske disclose processing of geometry and vertex data based on storage capacity. Storage is part of the shared resources in claim 3 and claim 4. The kind of data that is stored in the shared resources is irrelevant for meeting the limitation in claim 3 and 4. The claims do not require, e.g., that only pixel data is stored in the shared resources “...until enough shared resources become available and then use the shared resources to perform vertex calculation operations.” Ex. 1001 at Claim 3. Selzer + Fiske are capable of performing vertex operations followed by pixel operations, and vice versa, based on available storage capacity, as required by the claims.

166. Furthermore, Selzer + Fiske perform vertex or pixel calculation operations considering shared resources beyond storage. In Selzer, system busses transfer data to and from the framebuffer and the “master” module for distribution to the rendering modules. Ex. 1016 at 43-44. The bus in Selzer is a shared resource that affects processing of vertex and pixel operations. As Selzer notes, “in the case

of synchronizing....the master takes over system control and bypasses the pipeline with a direct access to the appropriate resource.” Ex. 1016 at 41. Thus, the master module may direct vertex or pixel data to a rendering module in order to load balance. Accordingly, the shared bus in Selzer teaches the limitation in claims 3 and 4.

167. Selzer also contains a digital signal processor (DSP). A POSITA would know that DSPs, like other processors, contain a memory management unit, bus control unit, core registers, floating-point and fixed-point ALUs, buses, and other circuitry within each processor. These resources are all “shared” for both vertex and pixel processing. Fiske also mentions a “processor pipeline” (Ex. 1010 at p. 210 and Figure 1) that has similar ALU units, buses, and circuitry as the one in Selzer. All of these shared resources are used in processing vertex and pixel data and fulfill the limitation in claims 3 and 4.

**B. Selzer + Fiske Disclose A Processor Unit That Performs Vertex Manipulation Operations And Pixel Manipulation Operations At Various Degrees Of Completion Based On Switching Between Instructions In The Instruction Store (Claim 11).**

168. As discussed below, it is my opinion that ATI’s arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. Fiske discloses performing operations “at various degrees of completion based on switching between instructions in the instruction store.”**

169. ATI states that Petitioner relies on multithreading and the thread priority system in Fiske to meet the limitation of claim 11: “Petitioner relies on Fiske’s thread priority system. Pet., 72” Response at p. 57.

170. As described before (¶¶144-147), Fiske discloses hardware context switches between threads. A POSITA understands that these context switches happen at various degrees of thread completion during a thread *stall*. In the multithreaded processor of Fiske, multiple threads can be executed using a single, shared processing core. At some point, the execution of an instruction in a thread *stalls* because of a dependency. For example, the instruction may require texture data that has to be fetched from texture memory. Texture access latency can be 100’s to 1000’s of cycles, so the execution resources during that amount of time are wasted. Only when the dependency is resolved can the thread continue execution and process the remaining instructions. In the case of a stall, the instruction of the thread is not completed, i.e., the instruction is at “various degrees of completion.” Note that this is consistent with my testimony “that the instructions that are constituent parts of the thread are the “operations” of claim 11. EX. 2123, 86:11-15” Response at p. 57.

171. At the point of a stall, Fiske teaches to switch to another thread. This is also known as ***context switching***. “Multiple-context processors provide register resources that allow rapid context switching between several threads as a means of tolerating long communication and synchronization latencies.” Fiske Abstract. “A context switch occurs when the processor switches from executing one loaded thread to executing another loaded thread, an operation that can be done in 1 to 20 cycles, depending on the processor design.” Ex. 1010 at p. 210.

172. During context switching, the multithreaded processor of Fiske wastes no processing resources. It completes the context switch “in 1 to 20 cycles,” which is much faster than waiting the 100s to 1000s of cycles during an instruction stall. Switching between threads in the multithreaded processor of Fiske is as simple as selecting a different thread for execution based on thread priority. The execution resources are not idle because another thread continues execution.

173. Therefore, the multithreading in Fiske discloses performing operations “at various degrees of completion based on switching between instructions in the instruction store” as required by claim 11.

**2. The combination of Selzer and Fiske discloses claim 11.**

174. ATI writes: “However, to the extent Fiske does disclose switching execution between unfinished threads, and further unfinished instructions, it

doesn't do so between vertex and pixel threads and instructions because it is indifferent to vertex and pixel operations; any purported switching occurs only between loaded high priority threads. EX. 2121, ¶377; EX. 1010, 211. Fiske doesn't discuss graphics processing at all. Id.”

175. As should be clear to ATI, Petitioner argues that the combination of Selzer and Fiske meets claim 11 and all other claim limitations of the '454 patent. Petitioner did not solely rely on Fiske. Thereby ATI's argument that Fiske does not discuss graphics processing is irrelevant.

**3. There is motivation to combine Selzer and Fiske for fast multithreaded graphics processing.**

176. ATI argues: “Moreover, even if Fiske disclosed what Petitioner alleges, there is no motivation to add that functionality to Selzer. EX. 2121, ¶378.” And: “For the rendering module to switch between rendering and geometry, it “consumes time.” Response at 48. Therefore, a POSITA would not add the ability to switch between unfinished pixel and vertex threads to Selzer, which would decrease Selzer's efficiency at processing graphics in the manner it deems most efficient and with highest performance, with no upside. EX. 2121, ¶378.” Response at p. 58.

177. As explained earlier, Fiske teaches that context switching between threads can be done very efficiently: “A context switch occurs when the processor

switches from executing one loaded thread to executing another loaded thread, an operation that can be done *in 1 to 20 cycles*, depending on the processor design.”

Ex. 1010 at p. 210. (emphasis added) A POSITA would understand that this context switching is faster than the 100s to 1000s of cycles to wait for a stall.

178. This efficient context switching between threads is one of the reasons that would motivate a POSITA to combine Selzer and Fiske in order to use the multithreaded architecture of Fiske with the flexible graphics processing system of Selzer. ATI’s statement: “the ability to switch between unfinished pixel and vertex threads to Selzer .... would decrease Selzer’s efficiency at processing graphics” is misguided and wrong.

**4. Selzer + Fiske disclose switching between incomplete instruction in the instruction store.**

179. ATI writes: “Separately, Petitioner hasn’t explained how Fiske (or Selzer) would switch between incomplete pixel and vertex operations “*based on* switching between instructions in the instruction store.” EX. 1001, Claim 11 (emphasis added). Petitioner merely asserts that switching between instructions of different threads meets the limitations of claim 11 but doesn’t describe how those instructions are located within an instruction store, which Petitioner alleges is Selzer’s master module. Pet., 74; EX. 2121, ¶379.” Response at p. 59

180. As explained earlier, the combination of Selzer and Fiske teaches fast context switching between threads when an instruction is stalled. That thread and that instruction are “at various degrees of completion.” It would be well understood by a POSITA that switching to another thread involves switching to a different instruction. That switch includes locating that instruction within an instruction store. As explained earlier, this is part of the instruction fetch in the processing pipeline.

181. Selzer teaches that all of its DSP processors contain an internal SRAM instruction store: “All modules contain a digital signal processor (DSP) with up to 256k \* 32 bit wide, fast static memory for instruction and data storage.” Selzer p. 40. The multithreading of Fiske, implemented on the processor of Selzer, would use this fast SRAM instruction store for context switches. Again, this is motivated by the observation of Fiske that thread context switches happen orders of magnitudes faster than thread stalls. A POSITA would know that the instructions of threads that are currently being processed would be stored in the internal SRAM memory of the DSPs in Selzer.

182. Therefore, the combination of Selzer + Fiske teaches “an instruction store” (the DSP SRAM for instructions), switching between “operations at various degrees of completion based on switching between instructions in the instruction

store” (thread context switches), and “vertex manipulation operations and pixel manipulation operations” (previously discussed) to meet the limitation of claim 11.

**C. Selzer + Fiske Disclose Execution of Instructions “In Response to” Receiving Selected Data (Claim 2).**

183. As discussed below, it is my opinion that ATI’s arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. ATIs interpretation of “in response to selected data from the general purpose register” is not supported by the evidence and goes counter to common usage by a POSITA.**

184. I have already addressed ATI’s improper reading of the claims in light of how a POSITA would understand the claims.

**2. Selzer demonstrates the execution of operations “in response to selected data from the general purpose register” (Claim 2).**

185. ATI falsely alleges that: “Petitioner doesn’t even allege that the execution by Selzer’s alleged processor unit is done in response to selected data.” Response at p. 59. This is false.

186. The operations the rendering module in Selzer may perform include generating the claimed vertex position and appearance data, which Selzer describes as transformation, clipping, polygon and patch subdivision, normal interpolation, and lighting operations. Ex. 1016 at 42. Selzer makes it clear that graphics

processing involves increasing numbers of graphics data: “In the entire system graphics data are processed simultaneous and transferred to the subsequent modules in parallel. From stage to stage the number of elements per object increases as the content of information per element decreases (Figure 4.5).” Ex. 1016 at p. 43 and Fig. 4.5. A POSITA recognizes this as the ***graphics processing pipeline***, explained in greater detail in my initial declaration (Ex. 1003, Section V.A.).

187. A POSITA knows, and Selzer explains, that data is transferred to subsequent modules in the graphics processing pipeline in response to data that has been previously processed. ““In the entire system ***graphics data are*** processed simultaneous and ***transferred to the subsequent modules*** in parallel.” Ex. 1016 at p. 43 and Fig. 4.5. (emphasis added) This requires the execution of operations (processing) “in response to” selected data as required by claim 2. For example, a triangle (geometry) may be rasterized into pixels, which are then blended into the framebuffer (pixel processing). The blending of those pixels happens “in response to” the preceding rasterization. There are countless stages within the graphics processing pipeline implemented by Selzer that process data in response to the output of the previous stage. Fig. 4.5 in Selzer shows some of those stages with their corresponding processing modules.

188. Selzer implements vertex and pixel processing of the graphics processing pipeline in the unified shader of the rendering module. “Because the rendering module can act as a geometry module too, it can also directly fetch graphics data from the master data FIFO and deliver processed data to the FIFOs of the appropriate rendering modules (see Section 4.5).” The vertex and pixel operations are performed based upon the respective vertex and pixel data and instructions that are selected from the registers under coordination of the master module. Ex. 1016 at p. 42. In addition, the rendering module generates a pixel color and conveys the calculated pixel colors and values back to the framebuffer. Id.

189. Therefore, Selzer discloses the graphics processing pipeline, and processes subsequent graphics data “in response to” data it created in prior stages. Selzer also discloses the “general purpose register block” of claim 2. Each of its rendering processors and the master module includes a block of registers (general register blocks) in the form of programmable memories and FIFOs that receive and maintain all geometry (vertex) and rendering (pixel) data. Ex. 1016 at 44-45. In addition, a distributed frame buffer is coupled to each rendering unit. During execution of the graphics processing pipeline, data is selected from the various

memories in the general purpose register. All of this demonstrates that “execution by Selzer’s alleged processor unit is done in response to selected data.”

190. It should be noted that the ID held that “performing rendering or geometry processing based on the data received” meets the claim requirement.

**3. Fiske demonstrates the execution of operations “in response to selected data from the general purpose register” (Claim 2).**

191. ATI writes: “Fiske confirms that it is the receipt and loading of threads, or groups of instructions, that cause execution—not receipt of data. “The software scheduler uses the priority to decide *which threads are loaded* . . . the hardware scheduler uses the priority to determine which loaded thread to execute.” EX. 1010, 211; EX. 2121, ¶341; EX. 2121, ¶381.” This is wrong.

192. As described earlier, Fiske disclose multiple thread contexts, which Fiske describes as storing the threads that are available for execution. The thread contexts described by Fiske include the data and instructions for execution. “Multiple register sets, including multiple instruction pointers, allow the state of multiple threads to be loaded and ready to run at the same time.” Ex. 1010 at p. 210. These thread contexts are stored in multiple registers (“general purpose register”) inside the multithreaded processor of Fiske. Fiske calls this a “multi-context processor.” Ex. 1010 at Title, p. 210, and Fig. 1.

193. Fiske teaches that these stored thread contexts are selected by a thread prioritization scheduling scheme and sent to a processor unit. “Thread prioritization is a simple means of guiding the scheduling of threads on a node. Each thread has a priority that indicates the importance of the thread in the overall problem. The software scheduler on each node chooses the highest priority threads as the loaded threads.” Ex. 1010 at p. 211.

194. Therefore, the processor in Fiske executed threads “in response to selected data” (e.g., based on the thread priority) from thread contexts that are stored in the “general purpose register” of the multithreaded processor. This meets the required elements of claim 2.

195. ATI further wrongly claims: “If the triggering event is the receipt of instructions, as it is in Selzer and Fiske, rather than receipt of data, the execution simply is not “in response to selected data,” it is simply on selected data, in response to receiving instructions. EX. 2121, ¶382.” Response at p. 60.

**D. Selzer + Fiske Disclose the “Sequencer,” “Instruction Store,” “Circuitry,” “Selection Circuit,” and “Arbiter” Within the “Unified Shader” As Required (Claims 2, 5-7, 10, 11).**

196. As discussed below, it is my opinion that ATI’s arguments are based on incorrect readings of the claims and misconceptions of the prior art and/or the technology in general.

**1. Selzer’s “Master Module” discloses a “Sequencer” and “Circuitry” (claims 2, 5, and 6)**

197. ATI writes: “Petitioner identifies Selzer’s “rendering module” as the claimed unified shader. Pet., 67, 75.” Response at p. 60. This ignores the remainder of the graphics architecture described in Selzer.

198. Selzer shows the basic module structure of his graphics architecture in Fig. 4.2, reproduced below.

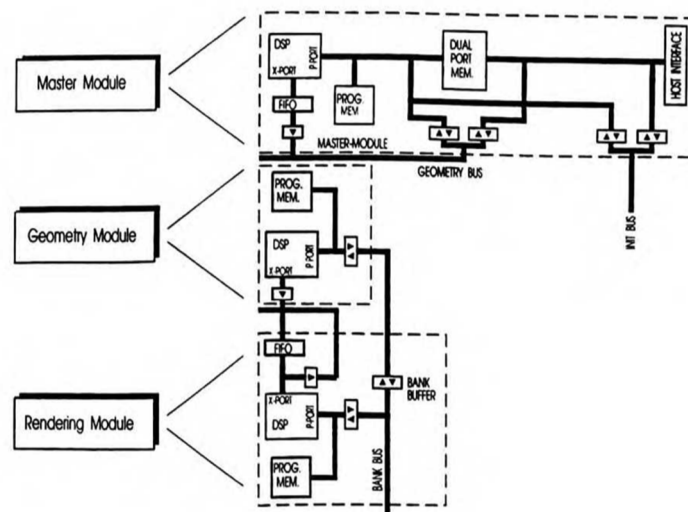


FIGURE 4.2. Basic module structure

ATI also reproduces this figure in their response. Response at p. 62. (with color annotations). Contrary to ATI’s assertion, the Master Module, Geometry Module, and Rendering Module in Selzer disclose the unified shader, sequencer, instruction store, circuitry, selection circuit, and arbiter. ATI is attempting to “draw boxes”

around the modules of Selzer without considering their functionality. For example, Selzer explains that its geometry and rendering modules could not be operational without the master module. “The master module is the systems supervisor, handles the communication to the host processor and is responsible for start-up and synchronising activities,” Ex. 1016 at 4.3.1 Overview “The master module traverses the graphics data structure and feeds graphics data to a special first-in-first-out memory (FIFO) for delivering to the appropriate processors.” Ex. 1016 at 4.3.1.1 Simply put, the other modules could not operate without the master module in Selzer. Similarly, the unified shader in the ‘454 patent could not operate without the arbiter (64), which is depicted outside of the unified shader (62) in Fig. 4A. Ex. 1001, Fig. 4A

199. As explained in my declaration, the Master Module in Selzer is the claimed Sequencer in the ‘454 patent. For example, the master module in Selzer accesses the graphics data in the system and “fetches the next primitive or task automatically.” Ex. 1016 at 43. The master module has access to various busses and the framebuffer and like the framebuffer maintains graphics data and its associated instructions to provide to the rendering module. All of the modules in Selzer contain storage for instructions and data. “All modules contain a digital signal processor (DSP) with up to 256k \* 32 bit wide, fast static memory for

instruction and data storage.” Ex. 1016 at p. 40. The Master Module can also directly access the graphics context of other modules. “In the case of synchronising or updating (e.g. graphics context, colour lookup tables, etc.) the master takes over system control and bypasses the pipeline with a direct access to the appropriate resource.” Ex. 1016 at p. 41. The master module has access to various busses and the framebuffer and like the framebuffer maintains graphics data and its associated instructions to provide to the rendering module. *Id.* Selzer describes the master module as the “systems supervisor” that “is responsible for start-up and synchronizing activities.” *Id.* at 40. The master module gathers the graphics data and tasks and supplies those tasks (data and instructions) to the appropriate processor unit. *Id.* at 41. As noted above, the master module is able to switch between sending vertex or pixel tasks to each rendering unit, allowing them to load balance among graphics operations. Once calculated, the data is stored in the local memories on each unit and eventually sent to the framebuffer. *Id.* The Master Module meets all of the requirements of claims 2, 5, and 6 of the ‘454 patent, including the claimed “circuitry operative to fetch data from a memory” Ex. 1001 at Claim 6.

**2. Selzer + Fiske Disclose a “Selection Circuit” and “Arbiter” (claims 7 and 10)**

200. ATI also wrongly claims that Selzer + Fiske does not disclose the claimed selection circuit and arbiter of claims 7 and 10 of the ‘454 patent. As explained in my initial declaration, the combination of Selzer + Fiske teaches that a circuit (e.g., master module, scheduling arbiter in Fiske) that provides information (i.e., data) to the store (i.e., memories and FIFOs) in response to a signal dictating whether to process vertex or pixel data. Because the modules in Selzer and Fiske are deciding what data to select based on the variable status of the system (e.g., the switching between vertex and pixel processing in the rendering modules in Selzer, or the context switching disclosed in Fiske), this process is performed by an arbiter. For example, the hardware scheduler in Fiske “chooses the loaded thread with the highest priority as the next thread using simple hardware, in order to minimize context switch overhead.” Ex. 1010 at 211. In this case, the “hardware scheduler” is the claimed arbiter, and the “simple hardware” within the arbiter provides the claimed control signal. Alternatively, Fiske discloses that the “software scheduler on each node chooses the highest priority threads as the loaded threads.” Ex. 1010 at 211. In this case, the “software scheduler” is the claimed arbiter, and instructions inside the software scheduler provide the claimed control signal. The threads in Fiske are vertex or pixel threads that fetch data from a cache,

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

**E. The “Sequencer” And “Instruction Store” in Selzer Maintain Instructions (Claims 2, 5, and 11)**

201. ATI wrongly asserts that claims 2, 5, and 11 in the ‘454 patent require that the sequencer is “maintaining instructions” or storing “instructions in the instruction store.” However, claims 2 and 5 clearly distinguish between “the sequencer maintaining instructions” and the instructions stored for processing by the processor unit: “...to **cause the processor unit to execute** vertex calculation and pixel calculation operations on selected data **maintained in the general purpose register block;**” Ex. 1001, claim 2 (emphasis added) “...to **cause the processor unit to execute** vertex calculation and pixel calculation operations on selected data **maintained in a store...**” Ex. 1001, claim 5 (emphasis added) “...wherein **the processor unit** of the unified shader performs the vertex manipulation operations and pixel manipulation operations at various degrees of

completion based on switching between **instructions in the instruction store.**”Ex. 1001, claim 11 (emphasis added) Claim 11 does not mention the sequencer.

202. Besides ATI’s wrong reading of the claims, the sequencer in Selzer has an instruction store that maintains instructions. As explained above, the Master Module in Selzer discloses storage for instructions. “All modules contain a digital signal processor (DSP) with up to 256k \* 32 bit wide, fast static memory for instruction and data storage.” Ex. 1016 at p. 40.

203. ATI mischaracterizes my deposition testimony regarding what the sequencer is in Selzer. Response 64. Regardless of what module it is in, I identified all of the components that together make the sequencer. I am not aware of any requirement that they must be in the same module.

204. Similarly, ATI is wrong in stating: “Petitioner further argues that “[t]he master module gathers the graphics data and tasks and supplies those tasks (data and instructions) to the appropriate processor unit.” Pet., 71-72. But that is incorrect. EX. 2121, ¶391. All Selzer says is that the master module puts data into the data FIFO.” Response at p. 64. As explained earlier, the master module can also directly access the graphics context of other modules. “In the case of synchronising or updating (e.g. graphics context, colour lookup tables, etc.) the

master takes over system control and bypasses the pipeline with a direct access to the appropriate resource.” Ex. 1016 at p. 41.

205. ATI also wrongly states: “But Petitioner doesn’t actually identify any component in Fiske as the claimed sequencer, and focuses on adding Fiske to Selzer’s rendering module, not the alleged sequencer, the master module. Id.” First, Selzer by itself discloses the claimed sequencer of the ‘454 patent as explained earlier. If Selzer is combined with Fiske, the multithreaded processing in the rendering module also discloses a claimed sequencer. As explained in my declaration, when the multiple thread contexts of Fiske are present in a rendering module of Selzer, the Fiske scheduling algorithm would choose the active thread for which the rendering module would execute the next instruction. That instruction for the particular thread would either be for a vertex calculation or for a pixel calculation. Because Fiske assigns a priority to each and every thread in the system, the scheduling and instruction selection may be very fine-grained and lead to an efficient and dynamic system that is able to move back and forth between vertex and pixel operations very rapidly.

**F. Selzer + Fiske Disclose the “Arbiter” (Claim 10) and “Control Signal” (Claims 7, 10)**

206. I have already addressed the issues raised by ATI for these claim limitations.

**G. Motivation to Combine Selzer + Fiske**

207. As discussed further below, a POSITA would find ample motivation to combine Selzer and Fiske. ATI's attacks ignore the teachings of both and the benefits associated with the combination.

**1. Both Fiske and Selzer adapt the activities of the processing units to the actual processing requirements**

208. ATI argues that a POSITA would not be motivated to combine Selzer and Fiske: "But Fiske literally says that it is not: "We do not consider dynamic load balancing issues in this paper." EX. 1010, 211; EX. 2121, ¶397." Response at p. 67. The fact that Fiske chose not to discuss dynamic load balancing in his paper (maybe for space reasons) does not mean that it is not a motivation to combine his teachings with Selzer.

209. Selzer defines dynamic load balancing as follows: "Nevertheless the size and the number of triangles an image consists of may vary from scene to scene or even from view to view. This will cause idle states within a fixed balanced architecture. With the intention of exploiting all the distributed computational power of the system, *the processing units have to be able to adapt their activities to the actual processing requirements* of the scene." Ex. 1016 at p. 46. (emphasis added)

210. Adapting the activities of the processing units to the actual processing requirements is enabled by the thread prioritization in Fiske: “When scheduling threads on such a processor, we must first decide which threads should have their state loaded into the multiple contexts, and second, which loaded threads to execute instructions at any given time. In this paper we show that both decisions are important, and that incorrect choices can lead to serious performance degradation.” Fiske Abstract. “Thread prioritization is a simple means of guiding the scheduling of threads on a node. ... The goal is to devote as many of the processor resources as possible to the tasks that are known to be critical to overall performance.” Ex. 1010 at p. 211. “In our case, the goal is to identify as exactly as possible a relative order in which threads should be run, and devote as many resources as possible to the most important threads.” Ex. 1010 at p. 211.

**2. The multithreading in Fiske improves the performance of rendering calculations in Selzer**

211. ATI argues: “Further, combining Fiske with Selzer because it would be antithetical to Selzer’s disclosure that its “peak performance” is when “all rendering processor power is exploited for rendering calculations and the rendering are working continuously.” EX. 1016, 48. It would make no sense to switch away from processing rendering calculations if such calculations were unfinished, as

Petitioner suggests is disclosed by Fiske. EX. 2121, ¶398. Doing so would lose the system's peak performance. Id.” Response at p. 67 This is wrong.

212. The multithreading of Fiske improves performance by several orders of magnitude. Fiske writes: “A thread swap involves swapping a loaded thread with an unloaded thread from the software queue. The cost of a thread swap is one to two orders of magnitude greater than a context switch, because it involves saving and restoring register state, and manipulating the thread scheduling queue.” Ex. 1010 at p. 210.

213. The reason, as explained earlier, is that the multithreaded processor in Fiske wastes no processing resources when a thread is stalled. First, the “multiple-context processor” in Fiske contains multiple storage locations (e.g., registers) to store multiple thread contexts. Fiske Figure 1. “Multiple-context processors provide register resources that allow rapid context switching between several threads as a means of tolerating long communication and synchronization latencies.” Ex. 1010 at Abstract.

214. These register resources improve performance during thread stalls: “Multiple register sets, including multiple instruction pointers, allow the state of multiple threads to be loaded and ready to run at the same time. Each time the currently executing thread misses in the cache or fails a synchronization test, the

processor can begin executing one of the other threads loaded in one of the other hardware contexts.” Ex. 1010 at p. 210. This performance improvement of multithreaded processors is well known to a POSITA, who would be motivated to combine it with the architecture in Selzer to improve graphics processing performance. ATI is wrong and misunderstands the main benefits of multithreading when writing: “It would make no sense to switch away from processing rendering calculations if such calculations were unfinished...” Response at p. 67. The opposite is true in a multithreaded graphics processor that combines Fiske and Selzer.

**3. Task switching in Selzer is not the same as multithreading in Fiske, but it helps motivate the combination of Selzer + Fiske**

215. ATI writes: “But Selzer already discloses an adequate task switching ability which maintains its peak performance. Id. Thus, adding Fiske solely for the purpose of switching threads, as opposed to tasks, is unnecessary and not something a POSITA would do. EX. 2121, ¶399.” Response at p. 67

216. Selzer describes task switching as follows: “The capability of automatically distributing the work load between geometry and rendering modules means inherently task switching between two jobs within the same application. Supported by the large local memory (up to 256k \* 32bit) the switching is reduced

to the saving and restoring of all processor registers, processing interrupt control and inspecting the rendering FIFO status.” Ex. 1016 at p. 47-48. And: “How much calculation time is eaten up by task switching and by which factor geometry calculations are accelerated if a rendering processor switches to geometry processing is evaluated in this chapter.” Ex. 1016 at p. 48.

217. Selzer discloses that its processor can switch between “jobs within the same application,” where a job may include rendering (pixel) calculations or geometry (vertex) calculations. Since Selzer does not disclose the ability to store multiple thread contexts within its processor, it is likely that the DSP in Selzer uses single-threaded (and not multithreaded) execution.

218. Selzer discloses “task switching,” which is similar to context switching, and “saving and restoring of all processor registers,” which is similar to hardware context registers in a multithreaded processor. ATI is wrong in claiming: “...adding Fiske solely for the purpose of switching threads, as opposed to tasks, is unnecessary and not something a POSITA would do.” On the contrary, the similarity between task switching (Selzer) and multithreading (Fiske) would be great motivation to combine Selzer and Fiske. This combination would be easy to accomplish and lead to orders of magnitude in performance increase.

**4. Combining Selzer and Fiske would be obvious and easy for a POSITA.**

219. ATI writes: “Given the lack of details provided by Selzer and Fiske, Petitioner hasn’t shown that a POSITA would have a reasonable expectation of success in combining them. Id.”

220. As discussed in the previous section, there is great similarity between the task switching DSP processor in Selzer and the multithreading processor of Fiske. Both processors contain internal local storage for data and instructions: “All modules contain a digital signal processor (DSP) with up to 256k \* 32 bit wide, fast static memory for instruction and data storage.” Ex. 1016 at p. 40. “Each processing node has a cache and some portion of the global memory.” Ex. 1010 at p. 212.

221. Selzer uses this local storage for task switching: “Supported by the large local memory (up to 256k \* 32bit) the switching is reduced to the saving and restoring of all processor registers, processing interrupt control and inspecting the rendering FIFO status.” Ex. 1016 at p. 47-48. Similarly, Fiske uses multiple context registers for multithreading: “Multiple-context processors provide register resources that allow rapid context switching between several threads as a means of tolerating long communication and synchronization latencies.” Ex. 1010 at Abstract and Fig. 1.

222. A POSITA would immediately recognize that one possible combination of Selzer and Fiske is to replace the processor of Selzer (DSP) with the multithreaded processor in Fiske. Since Selzer uses a DSP, it would make sense to use a multithreaded DSP processor to accomplish this combination. It is likely that a DSP processor manufacturer offers single- and multi-threaded versions of the same DSP architecture with the same pinout to make processor upgrades easy. In any case, it would merely be a matter of swapping the processors and upgrading the software without any additional changes to the graphics system in Selzer.

#### **H. Secondary Considerations**

223. ATI argues: “ATI’s unified shader was greeted with skepticism.” What ATI’s retelling of history misses is the fact that the unified shader was neither popularized or productized for PCs by ATI.

224. The unified shader architecture first became prominent with the release of DirectX 10 by Microsoft in 2006. This new DirectX API (Application Programming Interface) was a key feature of the Windows Vista operating system. DirectX 10 introduced Shader Model 4.0, which was designed to support unified shaders.

<https://en.wikipedia.org/wiki/DirectX> and

[https://en.wikipedia.org/wiki/Unified\\_shader\\_model](https://en.wikipedia.org/wiki/Unified_shader_model)

225. NVIDIA was one of the first major GPU manufacturers to adopt this technology with its GeForce 8 series, launched in November 2006. The NVIDIA GeForce 8800 GTX was particularly notable for being the first commercial GPU to use unified shaders. NVIDIA's implementation allowed for more flexibility in processing graphical data, leading to better performance and more realistic graphics. The Lindholm patents were invented and developed by Nvidia.

[https://en.wikipedia.org/wiki/GeForce\\_8\\_series](https://en.wikipedia.org/wiki/GeForce_8_series) and  
<https://www.hardwarezone.com.sg/feature-nvidia-geforce-8800-gtx-gts-g80-worlds-first-dx10-gpu/embracing-unified-shader-architecture>

226. ATI was late to introduce a PC graphics card with unified shader. It transitioned to unified shaders with its TeraScale architecture, introduced in the Radeon HD 2000 series in 2007, several months after the introduction of Microsoft's DirectX 10 API and the NVIDIA GeForce 8800 GTX graphics card. The ATI Xenos GPU was only used in the Xbox 360 video game console.

[https://en.wikipedia.org/wiki/Radeon\\_HD\\_2000\\_series](https://en.wikipedia.org/wiki/Radeon_HD_2000_series)

227. ATI writes: "the "fresh perspective" of ATI "force[d]" those in the industry to "reconsider" the graphics pipeline. EX. 2124, 1-2. ATI engineers themselves also reported experiencing some unexpected results during

development. EX. 2121, ¶417; EX. 2144, 5; EX. 2147 (Wasson), 1.” Response at p. 70.

228. However, the unified shader was introduced by Microsoft’s DirectX 10 API and popularized on PCs by NVIDIA, not ATI. ATI did not offer a “fresh perspective” nor should the results have been “unexpected” during development.

## **V. DECLARATION**

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

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

Patent No. 8,760,454  
Petition for *Inter Partes* Review  
Ex. 1018 – Declaration of Dr. Hanspeter Pfister

Executed this 20th day of May, 2024, at 6 Pioneer Rd., Arlington, MA  
02474, USA.

Respectfully submitted,

A handwritten signature in cursive script that reads "Hanspeter Pfister". The signature is written in dark ink and is positioned above a horizontal line.

---

Hanspeter Pfister, Ph.D.