

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

NVIDIA CORPORATION,
Petitioner,

v.

NEURAL AI, LLC,
Patent Owner.

Case IPR2025-00606
Patent 8,648,867

PATENT OWNER'S PRELIMINARY RESPONSE

TABLE OF CONTENTS

	Page
I. INTRODUCTION	1
II. PETITIONER FAILS TO DEMONSTRATE OBVIOUSNESS OVER ANY GROUND (ALL CLAIMS, ALL GROUNDS)	4
A. Petitioner Fails To Demonstrate That A POSITA Would Be Motivated Or Able To Combine Nickolls/GPU Gems Or Kirk/GPU Gems With A Reasonable Expectation Of Success (All Claims, All Grounds).	4
1. Petitioner’s Motivation To Combine Is Legally Insufficient Because It Is Founded Upon A False Premise.....	5
a. Petitioner’s Motivation Is Rooted In The Premise That GPU Gems’s Listing 31-2 Discloses Pointer Swapping ..	5
b. Petitioner Presents No Evidence That GPU Gems’s Listing 31-2 Discloses Pointer Swapping.	9
c. There Is No Reason To Assume That Listing 31-2 Discloses Pointer Swapping.	15
2. Petitioner Fails To Demonstrate A POSITA Would Be Motivated Or Able To Combine Nickolls/GPU Gems With A Reasonable Expectation Of Success.	22
a. Petitioner Fails To Show That Nickolls Thread Based System Would Benefit From Pointer Swapping.	22
b. Petitioner Fails To Explain How Pointer Swapping Could Be Implemented In Nickolls—Or How A POSITA Could Do So With Any Reasonable Expectation Of Success...	27
c. Petitioner Further Fails To Demonstrate A POSITA Would Be Motivated Or Able To Add “Numerical	

	Simulation” To Nickolls With A Reasonable Likelihood Of Success.	30
3.	Petitioner Fails To Demonstrate A POSITA Would Be Motivated Or Able To Combine Kirk/GPU Gems With A Reasonable Expectation Of Success.	36
a.	Petitioner Fails To Demonstrate That Kirk’s System Would Benefit From Pointer Swapping.	36
b.	Petitioner Fails To Explain How Pointer Swapping Could Be Implemented In Kirk, Much Less With A Reasonable Expectation Of Success.	39
c.	Petitioner Fails To Demonstrate A POSITA Would Be Motivated Or Able To Add GPU Gems’s “Numerical Simulation” To Kirk’s System With A Reasonable Expectation Of Success.	41
B.	Petitioner Fails To Show The Combinations Satisfy Numerous Limitations (All Claims, All Grounds).	45
1.	Petitioner Fails To Show That GPU Gems Discloses Or Renders Obvious Swapping Pointers (Claims 1-8, 16-19, All Grounds).	45
2.	Petitioner’s Inconsistent Identification Of The “First Pointer” Violates The Antecedent Basis Rule (Claims 16-19).	48
3.	Petitioner’s Mapping Of “The First Pointer” And “The Second Pointer” Is Unclear And Fails Under Either Colorable Interpretation (Claims 1-8).....	53
4.	Petitioner Fails To Show That Any Ground Discloses Or Renders Obvious Claim 9.	57
a.	Petitioner’s Analysis Fails To Account For The Material Differences In Claim 9.	58

b. Petitioner Fails to Show That Either Combination
Discloses Or Renders Obvious Any Of The Required
“Streams.”63

III. CONCLUSION.....65

TABLE OF AUTHORITIES

Page(s)

COURT DECISIONS

<i>3M Innovative Props. Co. v. Avery Dennison Corp.</i> , 350 F.3d 1365 (Fed. Cir. 2005)	56
<i>Becton, Dickinson & Co. v. Tyco Healthcare Group, LP</i> , 616 F.3d 1249 (Fed. Cir. 2010)	55, 63
<i>Corephotonics, Ltd. v. Apple Inc.</i> , 84 F.4th 990 (Fed. Cir. 2023)	57, 64
<i>In re Fritch</i> , 972 F.2d 1260 (Fed. Cir. 1992)	9
<i>Intelligent Bio-Sys., Inc. v. Illumina Cambridge Ltd.</i> , 821 F.3d 1359 (Fed. Cir. 2016)	14
<i>KSR Int’l Co. v. Teleflex Inc.</i> , 550 U.S. 398 (2007)	13, 64
<i>In re Magnum Oil Tools Int’l, Ltd.</i> , 829 F.3d 1364 (Fed. Cir. 2016)	13, 64
<i>Microsoft Corp v. Biscotti, Inc.</i> , 878 F.3d 1052 (Fed. Cir. 2017)	60
<i>Netflix, Inc. v. DivX, LLC</i> , 84 F.4th 1371 (Fed. Cir. 2023)	45, 57, 64
<i>In re Ratti</i> , 270 F.2d 810 (CCPA 1959)	44
<i>In re Schweickert</i> , 676 Fed. App’x. 988 (Fed. Cir. 2017)	26

<i>Unigene Labs., Inc. v. Apotex, Inc.</i> , 655 F.3d 1352 (Fed. Cir. 2011)	44
<i>Virtek Vision Int’l ULC v. Assembly Guidance Sys., Inc.</i> , 97 F.4th 882 (Fed. Cir. 2024)	26
<i>Wi-LAN, Inc. v. Apple Inc.</i> , 811 F.3d 455 (Fed. Cir. 2016)	49

AGENCY DECISIONS

<i>Apple Inc. v. Cellular Comm’ns Equipment, LLC</i> , IPR2015-00576, Paper 7 (June 12, 2015).....	27
<i>Apple Inc. v. Contentguard Holdings, Inc.</i> , IPR2015-00450 (June 29, 2015).....	60, 64
<i>Hulu LLC v. Sound View Innovations LLC</i> , IPR2018-00582, Paper 34 (Aug. 5, 2019) (informative)	26
<i>Phison Elecs. Corp. v. Vervain, LLC</i> , PGR2024-00048, Paper 9 (Apr. 28, 2025)	53
<i>R.J. Reynolds Vapor Co. v. Fontem Holdings I B.V.</i> , IPR2016-01692, Paper 45 (Mar. 2, 2018)	27
<i>Whatfix, Inc. v. WalkMe Ltd.</i> , IPR2024-00065, Paper 34 (Mar. 28, 2025)	53
<i>Xerox Corp. v. Bytemark, Inc.</i> , IPR2022-00624, Paper 9 (Aug. 24, 2022) (precedential)	13, 41, 64

STATUTES

35 U.S.C. § 312(a)(3).....	14, 56, 62, 64
35 U.S.C. § 314(a)	65

REGULATIONS

37 C.F.R. § 42.6(a)(3).....12

37 C.F.R. § 42.65(a)..... 13, 41

OTHER AUTHORITIES

Consolidated Trial Practice Guide
(Nov. 2019).....14

EXHIBIT LIST	
2001	Lowenstein Declaration In Support of Notice of Intent
2002	Woo Declaration In Support of Notice of Intent
2003	Declaration of Dr. Massimiliano Versace in Support of Patent Owner's Discretionary Denial Brief [Versace-Decl.]
2004	Email from Massimiliano Versace (Monday, June 26, 2017 1:58:15 PM) [Email-Chain]
2005	Neurala_presentation_nVidia_6_2017.pdf (Redacted) [Neurala-Presentation]
2006	First Amended Complaint For Patent Infringement in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.), Dkt. 30, December 12, 2024 [Amended-Complaint]
2007	Defendant's Answer To Plaintiff's Amended Complaint in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.), Dkt. 31, December 23, 2024 [Amended-Answer]
2008	Scheduling Order in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.), Dkt. 39, January 27, 2025 [Scheduling-Order]
2009	Defendant NVIDIA Corporation's Identification of Preliminary Claim Constructions served April 15, 2025 in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.) [NVIDIA-Preliminary-Claim-Constructions]
2010	Defendant's NVIDIA Corporation's Opening Claim Construction Brief in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.), Dkt. 71 [NVIDIA-Claim-Construction-Brief]

2011	Defendant NVIDIA Corporation’s Preliminary Invalidity Contentions served April 1, 2025 in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.) (Redacted) [Invalidity-Contentions-Cover]
2012	Ex. A8 – Invalidity of U.S. Patent No. 8,648,867 – Nickolls served April 1, 2025 in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.) [Invalidity-Contentions-Nickolls]
2013	Ex. A9 – Invalidity of U.S. Patent No. 8,648,867 – Kirk served April 1, 2025 in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.) [Invalidity-Contentions-Kirk]
2014	Ex. A11 – Invalidity of U.S. Patent No. 8,648,867 – GPU Gems served April 1, 2025 in <i>Neural AI, LLC v. NVIDIA Corp.</i> , No. 7:24-cv-00221 (W.D. Tex.) [Invalidity-Contentions-GPU-Gems]
2015	Reserved
2016	Reserved
2017	U.S. Patent No. 8,922,566 [NVIDIA-’566-patent]
2018	Excerpt of File History for U.S. Patent No. 8,922,566 [NVIDIA-’566-FH]
2019	Excerpt of File History for U.S. Patent Application No. 13/335,850 [13/335,850-FH]
2020	The Real Brains Behind the Artificial Intelligence, NEURALA, https://www.neurala.com/about-may (last visited June 14, 2025) [Neurala-About]
2021	Nefi Alarcon, “Inception Spotlight: AI Startup Neurala Sees 7X Speedup with NGC,” NVIDIA DEVELOPER (Sept. 25, 2019), https://developer.nvidia.com/blog/inception-spotlight-ai-startup-neurala-sees-7x-speedup-with-ngc/ [NVIDIA-Blog-Post]

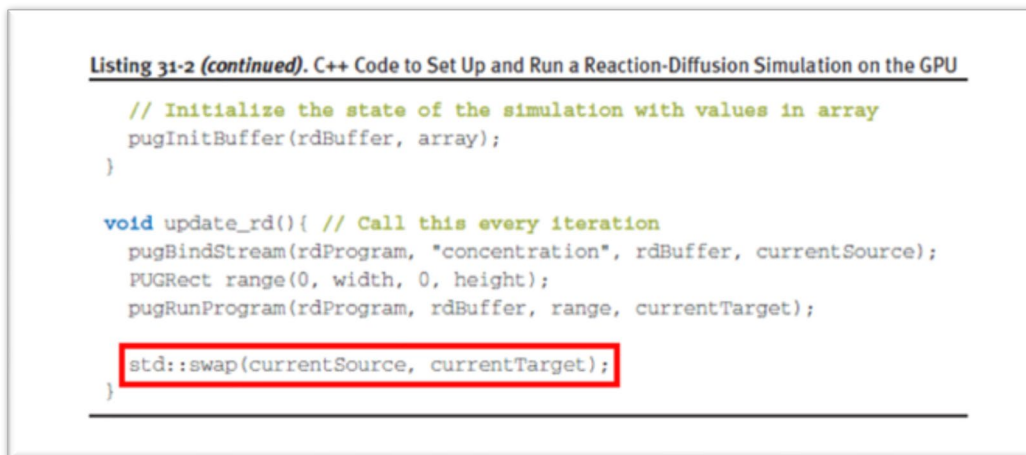
2022	Murali Gopalakrishna, LINKEDIN, https://www.linkedin.com/in/muralimohang/ (last visited June 12, 2025) [Gopalakrishna-LinkedIn]
2023	Kashi Anoosheh, LINKEDIN, https://www.linkedin.com/in/kashianoosheh/ (last visited June 12, 2025) [Anoosheh-LinkedIn]
2024	Jeff Herbst, GFT VENTURES, https://www.gft.vc/jeff-herbst (last visited June 14, 2025) [Herbst-Bio]
2025	Standing Order Governing Proceedings (OGP) 4.4—Patent Cases, W.D. Tex (Jan. 23, 2024) [Standing-Order-Governing-Proceedings]
2026	Screenshot of Time To Trial – Alan D. Albright, DOCKETNAVIGATOR (last visited June 13, 2025) [Albright-Time-To-Trial]
2027	Reserved
2028	Reserved
2029	Reserved
2030	Declaration of Sunil P. Khatri, Ph.D. [Khatri-Decl.]
2031	<i>File from CD Accompanying GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005): gpgpu_reactiondiffusion.cpp</i> (available at https://developer.nvidia.com/gpugems/gpugems2/inside-front-cover) [gpgpu_reactiondiffusion.cpp]
2032	Bjarne Stroustrup, <i>The C++ Programming Language</i> (3d ed. 1997) [The-C++-Programming-Language]
2033	Stanley B. Lippman, <i>C++ Primer</i> (5th ed. 2013) [Lippman]

2034	<i>Pointers</i> , cplusplus.com/doc/tutorial/pointers/ (last visited Apr. 23, 2025) [Pointer-Tutorial-cplusplus.com]
2035	<i>C++ Programming Language Pointers: References and Dynamic Memory Allocation</i> , https://personal.ntu.edu.sg/ehchua/programming/cpp/cp4_PointerReference.html (last visited Apr. 29, 2025) [NTG-University-Pointers]
2036	<i>C++ Pointers</i> , https://www.geeksforgeeks.org/cpp-pointers/ (last visited Apr. 21, 2025) [C++-Pointers-GeeksForGeeks]
2037	<i>C++ Pointers</i> , https://www.w3schools.com/cpp/cpp_pointers.asp (last visited Apr. 15, 2025) [W3 Schools - C++ Pointers.pdf]
2038	<i>File from CD Accompanying GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005): pug.h</i> (available at https://developer.nvidia.com/gpugems/gpugems2/inside-front-cover) [pug.h]
2039	ISO/IEC 14882:2003(E), Programming languages — C++ [INCITS]

I. INTRODUCTION

The Petition should be denied as it fails to demonstrate a reasonable likelihood of prevailing on any challenged claim. Indeed, it fails for multiple, independently sufficient reasons.

Petitioner presents two obviousness grounds—Nickolls/GPU Gems and Kirk/GPU Gems—each relying on GPU Gems. Both combinations hinge on a single line of code in GPU Gems Listing 31-2 (below) which Petitioner contends teaches “pointer swapping.” Petitioner’s premise is that two variables (`currentSource` and `currentTarget`) are pointers:

A screenshot of a code listing titled "Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU". The code is in C++ and shows a function `update_rd()` that binds a stream, runs a program, and then swaps two pointers, `currentSource` and `currentTarget`. The line `std::swap(currentSource, currentTarget);` is highlighted with a red rectangular box.

```
Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

Pet., 18, 34. From this premise, Petitioner argues that GPU Gems’s supposed pointer swapping would “benefit” Nickolls and Kirk by avoiding the need to copy large amounts of memory. *See* Section II.A.1(a).

But Petitioner fails to substantiate that premise. Neither Petitioner nor its

expert, Dr. Rosing, offers any evidence that `currentSource` and `currentTarget` are pointers. *See* Section II.A.1(b). Nor is there any reason to assume Listing 31-2 discloses pointer swapping. The code is written in C++, and in that language, pointers are typically declared with an asterisk. Petitioner presents no evidence that either `currentSource` or `currentTarget` is declared to be a pointer via an asterisk or through any other recognized means. Notably, in Listing 31-2 itself, `rdBuffer` and `rdProgram` are declared as pointers with an asterisk, but `currentSource` and `currentTarget` are not:

Listing 31-2. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU
See the source code on the accompanying CD for more details.

```
PUGBuffer *rdBuffer; PUGProgram *rdProgram;
```

In accord, the surrounding code in the CD attached to GPU Gems confirms that `rdBuffer` and `rdProgram` are declared as pointers with an asterisk and `currentSource` and `currentTarget` are not declared as pointers via an asterisk or any other recognized means:

```
28 | PUGBuffer    *g_rdBuffer = NULL;  
29 | PUGProgram   *g_rdProgram = NULL;  
30 |  
31 | PUGTarget    g_currentTarget = PUG_BACK;  
32 | PUGTarget    g_currentSource = PUG_FRONT;
```

Ex. 2031 [gpgpu_reactiondiffusion.cpp]; *see also* Section II.A.1(c).

But even if we were to assume, *arguendo*, that Listing 31-2 discloses pointer swapping, Petitioner still fails to demonstrate that a POSITA would be motivated to add it or GPU Gems’s computations to Nickolls or Kirk. The theory of motivation—avoiding costly memory copying—is unsubstantiated. Petitioner identifies no evidence that either Nickolls’s or Kirk’s system involves memory copying that pointer swapping could mitigate. Nor does Petitioner show that either combination could be implemented with a reasonable expectation of success, particularly in view of the substantial differences between GPU Gems on the one hand and Nickolls and Kirk, on the other. *See* Sections II.A.2 (Nickolls/GPU Gems) and II.A.3 (Kirk/GPU Gems).

And finally, even assuming the combinations could be made, Petitioner still fails to demonstrate that the resulting systems would satisfy several of the challenged claims’ limitations. *See* Section II.B.

For each of these reasons, institution should be denied.

II. PETITIONER FAILS TO DEMONSTRATE OBVIOUSNESS OVER ANY GROUND (ALL CLAIMS, ALL GROUNDS)

Petitioner's grounds should be rejected for multiple reasons. First, Petitioner fails to show that a POSITA would be motivated or able to combine Nickolls/GPU Gems or Kirk/GPU Gems with a reasonable expectation of success. *See* Section II.A. And even if the combinations were made, Petitioner fails to demonstrate that multiple limitations are met. *See* Section II.B.

A. Petitioner Fails To Demonstrate That A POSITA Would Be Motivated Or Able To Combine Nickolls/GPU Gems Or Kirk/GPU Gems With A Reasonable Expectation Of Success (All Claims, All Grounds).

As discussed below, Petitioner's combinations should be rejected because they are founded on the unsubstantiated and false premise that GPU Gems's Listing 31-2 discloses "pointer swapping." *See* Section II.A.1. But even if we assumed that GPU Gems's listing discloses pointer swapping, Petitioner fails to demonstrate that a POSITA would be motivated or able to combine Nickolls or Kirk with GPU Gems with a reasonable likelihood of success. *See* Section II.A.2-3.

1. Petitioner's Motivation To Combine Is Legally Insufficient Because It Is Founded Upon A False Premise.

Petitioner's entire motivation-to-combine rationale collapses under scrutiny because it rests on an incorrect premise. Petitioner contends a POSITA would be motivated to combine Nickolls and Kirk with GPU Gems's alleged disclosure of pointer swapping in Listing 31-2, which would allegedly benefit Nickolls and Kirk by avoiding the need to copy large chunks of memory. *See* Section II.A.1(a). Petitioner, however, fails—and, indeed, makes no serious attempt—to prove its premise: that the code it relies upon in GPU Gems actually discloses pointer swapping. *See* Section II.A.1(b). Even if that could be overlooked, GPU Gems's accompanying CD confirms that what Petitioner identifies as pointers are not, in fact, pointers. *See* Section II.A.1(c). Because Petitioner's combinations rest upon an unproven and false premise, they must be rejected.

a. *Petitioner's Motivation Is Rooted In The Premise That GPU Gems's Listing 31-2 Discloses Pointer Swapping.*

Petitioner contends that a POSITA would have been motivated to combine Nickolls with GPU Gems's alleged “swapping pointers” technique which purportedly provides “benefits ... while performing numerical computations”:

[A] POSITA would have understood the benefits of swapping pointers while performing numerical computations (as taught in GPU Gems) and been motivated to apply those techniques when writing programs

to perform numerical computations on the accelerator of Nickolls. A POSITA would have understood the combination of Nickolls and GPU Gems simply applied a known technique (*e.g.*, the numerical simulations and pointer-swapping of GPU Gems) to a known device (*e.g.*, the GPU-accelerated parallel processing system of Nickolls) ready for improvement to yield predictable results (*e.g.*, fast and efficient computation of complex numerical simulations).

Pet., 20-21¹ (citations omitted); 34 (“A POSITA implementing GPU Gems on the system of Nickolls would therefore include” pointer-swapping as taught in Listing 31-2). Petitioner’s argument for combining Kirk with GPU Gems is materially identical and likewise based on the premise that GPU Gems discloses pointer swapping which is allegedly beneficial while performing numerical computations. *See id.*, 57-58 (materially identical argument made for Kirk as was made for Nickolls (Pet., 20-21)); 67 (GPU Gems teaches that using the output of one computational cycle as an input for another “would be accomplished, in a system such as Kirk’s, by swapping the first pointer with the second pointer.”).

Petitioner contends that adding pointer swapping (per GPU Gems) to Nickolls or Kirk would beneficially “avoid[] copying large chunks of memory”:

¹ All emphasis in this brief is added unless otherwise noted.

It would also have been obvious to a POSITA to implement an algorithm on the hardware in Nickolls *in which the first and second pointers are swapped at the end of a computational cycle*. Because Nickolls describes using intermediate results as inputs to a new computational cycle (see *supra*), a POSITA would have been motivated to use known techniques to effectuate this efficiently. Ex1003, ¶174; Section III.A.3. *One such known technique was pointer swapping, which avoided copying large chunks of memory. Id.*

Pet., 34; *see also id.*, 67 (materially identical argument made for Kirk as was made for Nickolls (Pet., 34)); 29 (“By using pointers, developers can directly access and manipulate specific memory locations, allowing for efficient data retrieval and modification *without the need for copying large data sets.*”).

Petitioner contends that a single line of code in GPU Gems—“`std::swap` (`currentSource`, `currentTarget`);”—teaches pointer swapping:

Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);
    std::swap(currentSource, currentTarget);
}
```

Pet., 18 (“the code calls the ‘swap () ’ function, which swaps the pointer for the `currentSource` buffer with that of `currentTarget`, allowing the output of one computational cycle to become the input for the next.”).²

² While Petitioner points to other examples of alleged “pointer” swapping the GPU Gems textbook, Petitioner’s combination relies specifically upon the “chemical-diffusion reaction” of GPU Gems and its code in Listing 31-2, specifically. *See* Pet., 17-18, 19; *see also, e.g., id.*, 22, 23, 31, 32-33, 52, 56, 59. Further, certain of these other references cited (in Pet., 18) are disclosed as swaps of “textures” or “buffers,” not pointers as required. *See* Ex. 1006 [GPU Gems] 44 (it “swap[s] the roles of the two textures” not pointers); *id.*, 743 (“swap[s] the buffers” not pointers). While GPU Gems at 713 references a “swap [of] rho and newrho pointers,” Petitioner does not rely upon this swap in any of its grounds and for good reason. This swap does not fulfill many of the other requirements of the claim. For instance, Petitioner does not contend that rho or newrho references either the “first partition” of claim 16 into which “first input data” is transferred or “the second partition” of claim 1 that “store[s] second output data” “that becomes an input for a third computational cycle.”

But, as explained in the next section, Petitioner fails to demonstrate that the Listing 31-2 code it cites from GPU Gems actually discloses *pointer* swapping. This misunderstanding is not a minor flaw—it is fatal. The law is clear that “a misapprehension of the scope of [the prior art’s] teaching” does not support an obvious determination. *In re Fritch*, 972 F.2d 1260, 1265-66 (Fed. Cir. 1992).

b. *Petitioner Presents No Evidence That GPU Gems’s Listing 31-2 Discloses Pointer Swapping.*

As just discussed, Petitioner contends that a POSITA would be motivated to combine GPU Gems with Nickolls and Kirk based on GPU Gems’s alleged disclosure of “pointer swapping,” which Petitioner claims would beneficially avoid the need to copy large quantities of data. Petitioner concedes that neither Nickolls nor Kirk discloses “pointer swapping.” *See* Pet., 12 (“Nickolls does not explicitly provide the well-known technique of pointer swapping”), 33, 52 (“Kirk does not explicitly state that it uses the well-known technique of pointer swapping”); *see also* Ex. 1003 [Rosing-Decl.] ¶¶ 171, 309.

Instead, Petitioner contends that pointer swapping is disclosed by GPU Gems:

While Nickolls does not explicitly provide the well-known technique of pointer swapping as an example of how to do this, ***GPU Gems does. GPU Gems describes and provides code*** for a numerical simulation of a chemical reaction-diffusion on a GPU (such as that in Nickolls) ***that***

swaps pointers to input and output data as claimed. Thus, the combination of Nickolls and GPU Gems renders obvious all Challenged Claims.

Pet., 12-13; *see also, id.*, 17 (“GPU Gems describes the concept of swapping pointers of input and output data, as claimed in the ’867 patent.”); 33 (“swapping pointers to input and output data ... is disclosed in GPU Gems.”), 52 (“While Kirk does not explicitly state that it uses the well-known technique of pointer swapping, GPU Gems does.”).

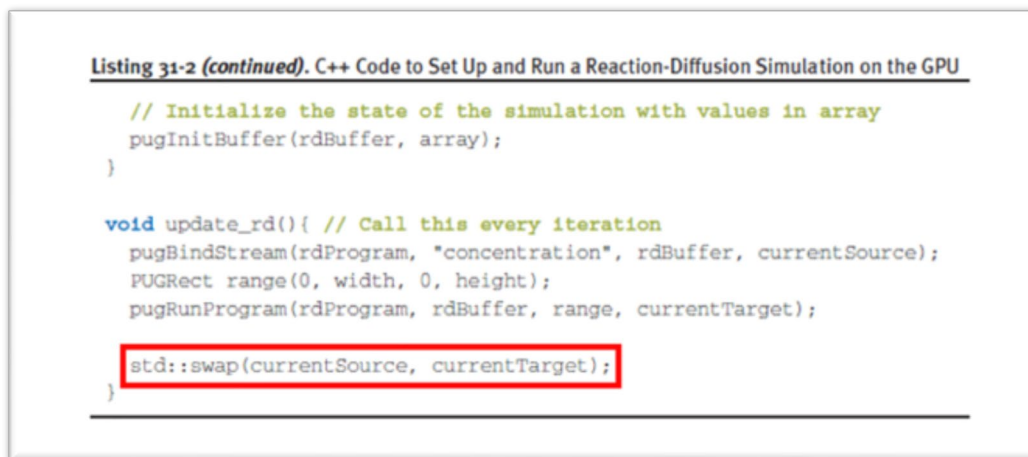
Specifically, Petitioner contends that a particular line of code in GPU Gems—“`std::swap (currentSource, currentTarget);`”—discloses swapping the “first pointer” (allegedly, `currentSource`) with the “second pointer” (allegedly, `currentTarget`):

```
std::swap(currentSource, currentTarget);
```

Pet., 34 (relying upon Ex. 1006 [GPU-Gems] 508), 33-34 (“GPU Gems describes a numerical simulation of a chemical reaction-diffusion on a GPU that swaps pointers to input and output data as claimed.”). Per Petitioner: “the code calls the ‘`swap()`’ function, which swaps the pointer for the `currentSource` buffer with that of `currentTarget`, allowing the output of one computational cycle to become the input for the next.” *Id.*, 18, *see also, id.*, 45, 52, 67, 77.

Petitioner, however, fails to demonstrate that either `currentSource` or `currentTarget` is a “pointer.” Indeed, it does not even try to prove this critical fact. Petitioner states:

At the end of each iteration, the code calls the “`swap()`” function, which swaps the pointers for the `currentSource` buffer with that of the `currentTarget` buffer, thus allowing the output of one computational cycle to become the input for the next computational cycle.



```
Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

Id., 508 (annotated). A POSITA implementing GPU Gems on the system of Nickolls would therefore include this claim step. Ex. 1003, ¶¶ 172-73.

Pet., 34. Thus, while Petitioner alleges that `currentSource` and `currentTarget` are pointers, it does not provide any evidence or explanation in support of its position. The Petition cites paragraphs 172-173 from Dr. Rosing’s

declaration (Ex. 1003), but these paragraphs are incorporated-by-reference and should not be considered. *See* 37 C.F.R. § 42.6(a)(3).

But even if those paragraphs were considered, Petitioner’s expert too does not establish that `currentSource` and `currentTarget` are pointers. In relevant part, Dr. Rosing states:

GPU Gems discloses a numerical simulation to be performed in parallel on a GPU wherein the GPUs read input from one buffer referenced by a pointer, write output to a second buffer referenced by a pointer, and swap pointers at the end of each computational cycle... Two buffers allow the simulation to alternate using one as input, one as output.”...[GPU Gems] discloses a function, “`void update_rd()`,” to be “call[ed] every iteration,” which supplies input and output pointers to the program performing the calculation, and then swaps the input and output pointers. *Id.*, 508. The iterating function “alternate[s]” buffers to be used as input and output by swapping pointers using the “`swap()`” function, as claimed in the ’867 patent.

Ex. 1003 [Rosing-Decl.] ¶ 172. Like Petitioner, Dr. Rosing *says* that GPU Gems discloses swapping pointers but fails to substantiate her position. She does not even clearly identify what the pointers are, nor does she explain *why* a POSITA would conclude that GPU Gems discloses pointer swapping.

In paragraph 164, Dr. Rosing more clearly states that `currentTarget` is a pointer. But there too, she merely asserts that this is so without evidence or

explanation:

At each iteration, [the program] calls the built-in function `pugRunProgram`, with... ***currentTarget***, a pointer to memory space allocated for the purpose of receiving output data.

Id., ¶ 164. Proclaiming `currentTarget` to be a pointer without proving it is quintessential *ipse dixit*.

It is well-settled that conclusory expert testimony such as this is entitled to little weight. *See* 37 C.F.R. § 42.65(a); *see also Xerox Corp. v. Bytemark, Inc.*, IPR2022-00624, Paper 9, 15–16 (Aug. 24, 2022) (precedential) (“conclusory and unsupported [expert testimony] adds little to the conclusory assertion for which it is offered to support, and is entitled to little weight.”). Nor can conclusory assertions—whether from a petitioner, its expert, or both—establish obviousness. *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 418–19 (2007) (“[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness.”) (citation omitted); *In re Magnum Oil Tools Int’l, Ltd.*, 829 F.3d 1364, 1380 (Fed. Cir. 2016) (“To satisfy its burden of proving obviousness, a petitioner cannot employ mere conclusory statements. The petitioner must instead articulate specific reasoning, based on evidence of record, to support the legal conclusion of obviousness.”).

Nor is this something that can be remedied later. The rules governing IPRs are clear. Petitioner must “make its case” in its petition, including presenting all evidence to make out a *prima facie* case of unpatentability. As the Federal Circuit has explained:

Unlike district court litigation—where parties have greater freedom to revise and develop their arguments over time and in response to newly discovered material—the expedited nature of IPRs bring with it an obligation for petitioners to make their case in their petition to institute. While the Board's requirements are strict ones, they are requirements of which petitioners are aware when they seek to institute an IPR.

Intelligent Bio-Sys., Inc. v. Illumina Cambridge Ltd., 821 F.3d 1359, 1369 (Fed. Cir. 2016); *see also* Consolidated Trial Practice Guide (Nov. 2019), 73 (“Petitioner may not submit new evidence or argument in reply that it could have presented earlier, *e.g.* to make out a *prima facie* case of unpatentability.”); 35 U.S.C. § 312(a)(3) (“the petition [must] identif[y], in writing and with particularity ... the evidence that supports the grounds ...”).

At bottom, Petitioner’s grounds rest on the premise that Nickolls and Kirk would benefit from the alleged pointer swapping taught in GPU Gems and that GPU Gems’s Listing 31-2 teaches a swap of pointers `currentSource` and `currentTarget`. Because Petitioner did not prove this foundational premise, and may not do so later, the Petition should be rejected.

c. *There Is No Reason To Assume That Listing 31-2
Discloses Pointer Swapping.*

As just demonstrated, Petitioner's combinations fail to launch because they rest upon an unproven premise; that `currentSource` and `currentTarget` are pointers. This is sufficient reason to reject the Petition. But, if any more were needed, there is (i) no reason to assume that `currentSource` and `currentTarget` are pointers and (ii) much reason to doubt that this is so. As explained below, the code in question is written in C++. In C++, a pointer must be declared and in that programming language a pointer is declared through established syntax, such as an asterisk (*). Petitioner presents no evidence that `currentSource` or `currentTarget` are declared as pointers via an asterisk or any other established syntax. What is more, a review of GPU Gems's broader source code confirms they are not declared as pointers.

As Dr. Khatri explains, the GPU Gems code in question is written in C++:

The GPU Gems code relied upon by Petitioner makes clear that it is written in the C++ programming language:

Listing 31-2 (continued). **C++ Code** to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

See Ex. 1006 [GPU-Gems] 508 (annotated).

Ex. 2030 [Khatri-Decl.] ¶ 44.

In C++, a pointer is declared via recognized syntax, usually an asterisk. As Dr. Khatri explains, this is confirmed by multiple references:

In C++, “[b]efore a name (identifier) can be used in a C++ program, it must be declared.” Ex. 2032 [The-C++-Programming-Language] 78. A pointer too must be declared. Pointers are typically declared by placing an asterisk (*) in front of the name. For instance, Stroustrup, the creator of the C++ language, explains in his textbook:

A declarator is composed of a name and optionally some declarator operators. The most common declarator operators are (§A.7.1):

*	<i>pointer</i>	<i>prefix</i>
---	----------------	---------------

Id., 80 (annotated).

Multiple references confirm that an asterisk (*) prefix may be used to declare a pointer. *See, e.g.*, Ex. 2033 [Lippman] 52 (“We define a pointer by writing a declarator of the form `*d`, where `d` is the name being defined. The `*` must be repeated for each pointer variable.”); Ex. 2034 [Pointer-Tutorial-cplusplus.com] 4 (“The declaration of pointers follows this syntax: `type * name`”); Ex. 2035 [NTG-University-Pointers] 2 (“Pointers must be declared before they can be used, just like a normal variable. The syntax of declaring a pointer is to place a `*` in front of the name. ... Take note that you need to place a `*` in front of each pointer variable ...”); Ex. 2036 [C++-Pointers-GeeksForGeeks] 1 (“A pointer can be declared in the same way as any other variable but with an asterisk symbol (`*`)”); *see also* Ex. 2037 [W3 Schools - C++ Pointers.pdf] 1 (“A pointer variable points to a data type (like `int` or `string`) of the same type, and is created with the `*` operator”).

Significantly, GPU Gems confirms that it does not deviate from this syntax. GPU Gems itself declares a pointer using an asterisk. For instance, earlier in Listing 31-2 itself, `rdBuffer` and `rdProgram` are declared as pointers, by preceding each identifier with an asterisk: `*rdBuffer` and `*rdProgram`.

Listing 31-2. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU
See the source code on the accompanying CD for more details.

```
PUGBuffer *rdBuffer; PUGProgram *rdProgram;
```

Ex. 1006 [GPU-Gems] 507 (annotated).

Ex. 2030 [Khatri-Decl.] ¶¶ 45-47.

Neither `currentSource` nor `currentTarget`, however, is declared as a pointer. Certainly, Petitioner and Dr. Rosing present no evidence that either is. And, in fact, a deeper review of the GPU Gems source code confirms they are not.

Per Dr. Khatri:

Nothing in the GPU Gems exhibit indicates that either `currentSource` or `currentTarget` is a pointer. No evidence is provided that either is declared with an asterisk (*) or any other syntax that might declare a pointer. Nor does a further examination of GPU Gems suggest that either is a pointer. The code Petitioner identifies in the GPU Gems textbook is only a portion of a larger source code file. Ex.1006 [GPU-Gems] 507 (“See the source code on the accompanying CD for more details.”). While in the code excerpt cited by Petitioner `currentSource` and `currentTarget` are never declared or initialized, this portion of the code can be found on the accompanying CD. Below is the point in the code that declares `currentSource` and `currentTarget`³:

³ While in the source code file on the CD, certain variable names are appended with “g_”, these variables correspond to their similarly-named counterparts lacking the “g_” in the sample code excerpt. The prefix “g” is a style

```
28 | PUGBuffer   *g_rdBuffer = NULL;  
29 | PUGProgram  *g_rdProgram = NULL;  
30 |  
31 | PUGTarget   g_currentTarget = PUG_BACK;  
32 | PUGTarget   g_currentSource = PUG_FRONT;
```

Ex. 2031 [gpgpu_reactiondiffusion.cpp] lines 28-32. The source code file, like the excerpt, is written in C++. As can be seen above, `g_rdBuffer` and `g_rdProgram` are declared as pointers using asterisks: `*g_rdBuffer` and `*g_rdProgram`. By contrast, the code declaring and initializing `g_currentSource` and `g_currentTarget` on lines 31 and 32 lack asterisks, or any other syntax that might declare a pointer, and thus `g_currentSource` and `g_currentTarget` are not pointers. In fact, they are an enumerated type (declared on lines 27-32 of the file `pug.h`, which is included on line 9 of Ex. 2031 [gpgpu_reactiondiffusion.cpp]). See Ex. 2038 [pug.h]. If they were pointers, one might expect the code to state, `*g_currentTarget` and `*g_currentSource`.

convention used to indicate global scope. See e.g., [<https://learn.microsoft.com/en-us/windows/win32/stg/coding-style-conventions>]. As can be seen, the added prefix is not unique to `currentSource` and `currentTarget`. `rdBuffer` and `rdProgram` have similarly become `g_rdBuffer` and `g_rdProgram`.

Ex. 2030 [Khatri -Decl.] ¶ 48.

In addition to `currentSource` and `currentTarget` not being pointers, the “`std::swap (currentSource, currentTarget)`” code that Petitioner relies upon does not swap pointers for another reason; it does not swap pointers, it swaps (exchanges) values. Per Dr. Khatri:

The “`std::swap`” function is a part of the C++ Standard Library. “`std`,” more specifically, refers to the standard C++ library and a “`swap`” function exchanges the value of two objects. Ex. 2039 [INCITS] 555 (“`swap`” “[e]xchanges values stored in two locations.”). Thus, the line of code Petitioner relies upon,

```
std::swap(currentSource, currentTarget);  
)
```

means that the value of `currentSource` and `currentTarget` are exchanged. To do this, the function may create a copy of `currentSource` and store it in a temporary variable, then assign the `currentTarget` to `currentSource`, and finally assign the temporary variable to `currentTarget`.

Copying values (which happen to be values of an enumerated type in this case), however, does not satisfy the claims which require swapping the “first pointer” and the “second pointer.” Rather, copying values (textures in particular) is the thing the ’867 patent sought to avoid by swapping pointers. *See, e.g.*, Ex. 1001 [’867] 11:38-42 (“Copying textures is an expensive operation, computationally, but

since the textures are referred to by texture IDs (pointers), swapping these pointers for input and output textures after each time step achieves the same result at a much lesser cost.”).

Ex. 2030 [Khatri -Decl.] ¶¶ 51-52.

In sum, Petitioner fails to prove that `currentTarget` and `currentSource` are pointers. Petitioner and Dr. Rosing provide only conclusory assurances that GPU Gems discloses pointer swapping but provide no evidence or explanation for why a POSITA would consider either `currentSource` or `currentTarget` to be pointers. This alone should prove fatal.

A review of GPU Gems’s underlying source code, moreover, confirms that neither is declared as a pointer, *e.g.*, by use of an asterisk. Rather, they are values within an enumerated type. Further, the code Petitioner relies upon, “`std::swap(currentSource, currentTarget)`” refers to an exchange of values, such that the values of `currentSource` and `currentTarget`, are copied and exchanged. In this regard, GPU Gems teaches value swapping, which the ’867 patent sought to avoid. This confirms that Petitioner’s proposed combinations rest upon a fatally false premise.

But, even if we were to ignore that GPU Gems’s Listing 31-2 fails to

disclose pointer swapping, Petitioner still fails to prove a POSITA would be motivated or able to add such pointer swapping to Nickolls and Kirk. *See* Sections II.A.2-3.

2. Petitioner Fails To Demonstrate A POSITA Would Be Motivated Or Able To Combine Nickolls/GPU Gems With A Reasonable Expectation Of Success.

Even assuming, *arguendo*, that GPU Gems's Listing 31-2 teaches the claimed pointer-swapping technique, Petitioner still fails to demonstrate that a POSITA would be motivated to incorporate that technique, or numerical simulations for that matter, into Nickolls's system. The combination Petitioner proposes solves no existing problem in Nickolls and offers no improvement to Nickolls's system. *See* Section II.A.2(a). Petitioner further fails to show that a POSITA could implement pointer swapping in Nickolls with a reasonable expectation of success. *See* Section II.A.2(b). Finally, Petitioner fails to prove a POSITA would be motivated or able to add GPU Gems's numerical simulation into Nickolls with a reasonable expectation of success. *See* Section II.A.2(c).

a. *Petitioner Fails To Show That Nickolls Thread Based System Would Benefit From Pointer Swapping.*

Petitioner contends a POSITA would be motivated to incorporate pointer swapping into Nickolls on the premise that pointer swapping alleviates the need to copy large amounts of information. Petitioner, however, has no explanation for

how Nickolls's cooperative thread array (CTA) based system would benefit from pointer swapping given that each of the threads can access shared memory without any need for copying or pointer swapping.

Petitioner, again, contends a POSITA would be motivated to incorporate GPU Gems's (alleged) pointer swapping into Nickolls's system. *See* Section II.A.1(a)-(b); *see also* Pet., 12-13, 21. Petitioner further argues that a POSITA would be motivated to employ pointer swapping in Nickolls to use intermediate results as inputs to a new computational cycle to “avoid[] copying large chunks of memory.” *Id.*, 34-35; *see also id.*, 45.

Petitioner, however, fails to identify any copying-related inefficiency in Nickolls that pointer swapping would solve or any other benefit that pointer swapping would add to Nickolls. As Dr. Khatri explains, Nickolls already avoids the need to copy large amounts of data through a system that utilizes cooperative parallel threads that can access intermediate results from shared memory that is accessible by all threads in the group:

Nickolls discloses a system for parallel data processing using groups of threads called cooperative thread arrays (“CTAs”), in which each thread is assigned a unique identifier (“thread ID”) which is “assigned at thread launch time”. Ex. 1004 [Nickolls] Abstract. Threads in a CTA operate concurrently and are designed to share intermediate results using shared memory—*i.e.*, memory that is

simultaneously accessible by all threads in the group. *See id.*, 8:10-14 (“Threads in a CTA can share intermediate results with other threads in the same CTA ***using a shared memory that is accessible to all of the threads ...***”), 20:46–51 (“Instructions to access shared memory or shared global register file space advantageously identify locations to be read and/or written as a function of the thread ID. Where such an instruction is encountered, the thread reads its thread ID and computes the target address from the thread ID using the function specified in the CTA program.”); cl. 2 (“loading the input data set into a shared memory accessible to all of the plurality of threads.”).

Ex. 2030 [Khatri-Decl.] ¶ 56. Once data is written to shared memory, each thread can access the stored data without the need for copying or pointer swaps, but rather, by using target addresses that are computed from the thread ID:

As Nickolls explains, threads thus compute read and write addresses deterministically based on their thread IDs. *Id.*, 8:25-28 (“By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner.”). Specifically, the CTA program “includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID,” and also includes “an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID.” *Id.*, 8:15-17, 8:23–25. This allows data written by one thread to be accessed not only by that same thread, but also by any other

thread in the CTA—through shared memory and without requiring any copying between memory regions or pointer swapping between the threads of the CTA. *Id.*, 8:10-13 (“Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads”), 8:28-30 (“Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA.”). In this way, intermediate data may be accessed and reused from any thread with an appropriate address calculation, and without any copying or pointer swapping. In fact, pointer swapping is made unnecessary in this approach, and would reduce the efficiency of the computation.

Again, because all threads within Nickolls’s CTA have concurrent access to a common shared memory region, and addresses of each thread within that region are computed deterministically based on thread ID, any thread can read from and write to this common shared memory space using predictable addresses. Thus, intermediate results written by one thread may be retrieved not only by that same thread, but by any other thread in the CTA—all without copying data or swapping pointers. *See id.*, 8:10-30, 20:46–51.

In fact, “each thread in a CTA executes the same CTA program on a different subset of an input data set.” *Id.*, 8:4-6. The “CTA program includes an instruction to compute an address in shared memory to which particular data is to be written” or “from which data is to be read”, which allows the address computation to be done

simultaneously. *Id.*, 8:15-25; *see also id.*, 20:46-51 This efficiency is negated if pointers are swapped, since the different threads in the CTA would lose the parallel nature of the address update due to synchronization issues.

Ex. 2030 [Khatri-Decl.] ¶¶ 57-59.

Even if pointer swapping were assumed to be a known technique, a POSITA would see no application for it in Nickolls’s architecture which uses shared memory for all relevant intermediate results and enables memory access through fixed thread ID-derived addresses. Ex. 2030 [Khatri-Decl.] ¶ 60.

Without a memory-copying problem to solve, and with a system already designed to avoid one through alternative means, a POSITA would see no reason to pursue the combination Petitioner proposes. A POSITA is not motivated to add a feature that provides no benefit. *See Virtek Vision Int’l ULC v. Assembly Guidance Sys., Inc.*, 97 F.4th 882, 887 (Fed. Cir. 2024) (“[T]here must exist a motivation to combine various prior art references in order for a skilled artisan to make the claimed invention.”); *Hulu LLC v. Sound View Innovations LLC*, IPR2018-00582, Paper 34, 20-21(Aug. 5, 2019) (informative) (rejecting combination where not clear why it would be a “good idea” to modify); *In re Schweickert*, 676 Fed. App’x. 988, 995 (Fed. Cir. 2017) (remanding where unexplained how “Birrell would have ‘benefitted from the advantages of ...

Cunniff’s semaphore mechanism,”) (non-precedential). Nor would a POSITA be motivated to add a redundant feature. *R.J. Reynolds Vapor Co. v. Fontem Holdings I B.V.*, IPR2016-01692, Paper 45, 26 (Mar. 2, 2018) (denying institution, finding no motivation to combine; “Because [the first reference] already discloses a reliable means for transporting liquid from a liquid supply to a heating element, we do not find that one of ordinary skill in the art would have been motivated to use the technique of [the second reference] to improve [the first]”); *Apple Inc. v. Cellular Comm’ns Equipment, LLC*, IPR2015-00576, Paper 7, 12 (June 12, 2015) (denying institution where “Petitioner ... fails to provide a specific explanation why [the primary reference] did not already accomplish [the asserted benefits of the combination] by itself.”).

b. *Petitioner Fails To Explain How Pointer Swapping Could Be Implemented In Nickolls—Or How A POSITA Could Do So With Any Reasonable Expectation Of Success.*

Even assuming *arguendo* that the code in GPU Gems Listing 31-2 discloses pointer swapping (it does not) and that a POSITA might see some theoretical benefit to adding pointer swapping to Nickolls, Petitioner still fails to demonstrate that the proposed modification could be implemented in Nickolls with a reasonable expectation of success. Indeed, Petitioner has not accounted for the substantial changes that would be required, or why a POSITA would, could, or should attempt

such a thing.

Nickolls operates on a deterministic, thread-ID-based memory address calculation model: each thread computes explicit addresses for reads and writes using its thread ID. *See* Ex. 1004 [Nickolls] 8:10–30, 20:46–51; Ex. 2030 [Khatri-Decl.] ¶ 63. Threads compute, in parallel from the CTA program, explicit memory addresses based on their thread ID using fixed functions. Ex. 2030 [Khatri-Decl.] ¶ 63. This model enables shared-memory coordination without pointer swapping or copying. *Id.* In fact, the use of pointers would negate the performance of the Nickolls approach, because instead of a parallel address computation, the threads in the CTA would require synchronization if pointer swapping was involved. *Id.*, 64.

Even if POSITA did disadvantageously decide to use pointer swapping, Petitioner offers no account of how a POSITA could implement pointer swapping within the Nickolls model. As Dr. Khatri explains:

Implementing pointer swapping in Nickolls presumes the existence of pointer variables that can be reassigned across computational cycles to toggle inputs and outputs. But Nickolls contains no such constructs and no mechanisms to support them. Nickolls does not use pointers to reference and swap memory locations. In particular, the word “pointer” does not occur anywhere in Nickolls. *See generally* Ex. 1004 [Nickolls].

Attempting to implement pointer swapping into Nickolls would require substantial changes to its architecture, with degraded performance. This would include, among other things, defining pointer variables capable of being swapped; storing and updating those references; synchronizing the swaps across the different threads in the CTA; and rewriting memory address calculation logic for the threads in the CTA to depend on those variables. None of this is suggested by Nickolls or accounted for by Petitioner. Nor would all the effort yield improved results in a CTA system, because the inherent parallelism of thread computation would be lost with the synchronization issues that would be exposed by the pointer swaps.

Id., ¶ 66-67.

Additionally, no explanation is given for how pointer swapping would co-exist with Nickolls thread-based system:

Petitioner additionally fails to account for how pointer swapping could exist within Nickolls's thread-based architecture. As explained previously, Nickolls's parallel threads can all access shared memory without any need for pointer swapping. If pointer swapping were added, it's not even clear what that would mean. Would it override the thread-based model? Co-exist with it? How? Would the pointers just sit unused or used in some particular context? None of this is explained. Nor is it explained how the thread-based system would even interact with a pointer-swapping overlay. The combination of pointer swapping with the existing efficient parallel thread ID based address computation

of Nickolls seems to be a hindsight-based construction by Petitioner, to meet the claim elements of the '867 patent.

Id., ¶ 68. In this regard, Petitioner's combination is a bit like adding a diesel engine to a car that already has a gasoline engine. It adds nothing, detracts from what is already there, you wouldn't be motivated to do it, you wouldn't know how to do it, and you wouldn't even understand what it would *mean* to do it. Saying that a diesel engine is "known" does not address the fundamental problems bedeviling this combination.

Ultimately, Petitioner never explains *how* pointer swapping could even function in Nickolls. The Petition simply assumes, without explanation, that Nickolls's architecture could support pointer reassignment. Pet., 32-34. But the record offers no basis to conclude that Nickolls's architecture is compatible with pointer swapping, let alone that a POSITA could reasonably expect to implement such a modification with success.

c. *Petitioner Further Fails To Demonstrate A POSITA Would Be Motivated Or Able To Add "Numerical Simulation" To Nickolls With A Reasonable Likelihood Of Success.*

Petitioner further fails to demonstrate that a POSITA would be motivated or able to implement GPU Gems's numerical simulations within Nickolls's system,

much less with a reasonable expectation of success.⁴ Petitioner fails to account for the material differences between GPU Gems's and Nickolls's systems. While GPU Gems's Listing 31-2 does not use pointer swapping, it does use the `rdBuffer` pointer to identify where input data is stored. Nickolls's system, by contrast, has an entirely different, non-pointer-based system for memory access which uses thread IDs. Petitioner does not explain why a POSITA would be motivated or able to implement GPU Gems's computation (which is reliant upon the `rdBuffer` pointer) in Nickolls (which is reliant upon non-pointer, thread ID-based addressing) with a reasonable expectation of success.

⁴ Petitioner's contention that a POSITA would be motivated to implement GPU Gems's numerical simulations into Nickolls is directly tied to its errant arguments concerning pointer swapping. *See* Pet., 20-21 ("a POSITA would have understood the benefits of swapping pointers while performing numerical computations (as taught in GPU Gems) and been motivated to apply those techniques when writing programs to perform numerical computations on the accelerator of Nickolls."). Consequently, Petitioner's arguments concerning implementing numerical computations in Nickolls fail for the additional reasons set forth in Sections II.A.2(a)-(b).

As previously discussed, Petitioner fails to prove that `currentSource` or `currentTarget` are pointers, much less pointers that are swapped. *See* Section II.A.1. While that critical linchpin of Petitioner's case is not shown, that does not mean that pointers play no role in the GPU Gems Listing 31-2 simulation. To the contrary, and by way of example, at the outset of Listing 31-2, `rdBuffer` is declared as a pointer via an asterisk:

Listing 31-2. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU
See the source code on the accompanying CD for more details.

```
PUGBuffer *rdBuffer; PUGProgram *rdProgram;
```

Ex. 1006 [GPU Gems] 507; *see also* Ex. 2030 [Khatri-Decl.] ¶ 47.

As Dr. Khatri explains, in Listing 31-2 `rdbuffer` is a pointer to the memory address where input data is stored:

Listing 31-2 runs an iterative numerical simulation. The simulation models how the concentrations of two or more reactants in a solution evolve as they undergo the process of chemical reaction and diffusion. Ex. 1006 [GPU Gems] 505.

To run the iterative simulation, and to access the input data across iterations, Listing 31-2 uses a pointer, `rdbuffer`, to identify where the input data needed for the simulation's equations are stored. This is reflected below in Listing 31-2 which shows that the `pugAllocateBuffer()` function returns the `rdBuffer` pointer:

Listing 31-2. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU
See the source code on the accompanying CD for more details.

```
PUGBuffer *rdBuffer; PUGProgram *rdProgram;

void init_rd(){ // Call this once.
    pugInit();    // Start up the GPU framework

    // Create a "double-buffered" PUGBuffer. Two buffers allow the
    // simulation to alternate using one as input, one as output
    PUGBuffer *rdBuffer = pugAllocateBuffer(width, height,
                                             PUG_READWRITE, 4, true);

    PUGProgram *rdProgram = pugLoadProgram("rd.cg", "rd");
```

Ex. 1006 [GPU Gems] 507 (annotated), 504. The `rdbuffer` pointer is utilized in the function `pugBindStream()`. As seen below, this function is used in every iteration to identify where the input data needed for the simulation is stored.

Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
    // Initialize the state of the simulation with values in array
    pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

Id., 508. Thus, while no pointers are shown as being swapped in Listing 31-2, the computations do rely upon the `rdBuffer` pointer for memory storage and retrieval.

Ex. 2030 [Khatri-Decl.] ¶¶ 73-74.

While GPU Gems's computations rely upon the `rdBuffer` pointer to identify where input data is stored, that system is vastly different from (and incompatible with) Nickolls, who uses a completely different non-pointer-based system for memory access which uses thread IDs. As previously explained, there is no teaching or disclosure of pointers in Nickolls and Nickolls's thread-based system has no role for pointers and no benefit from them. *See* Section II.A.2(b); Ex. 2030 [Khatri-Decl.] ¶ 76. Nickolls instead discloses a cooperative thread array (CTA) based system wherein threads may access information from shared memory based on an address computation that is based on their thread ID. *See id.*

Petitioner provides no explanation for how GPU Gems's simulation reliant upon the `rdBuffer` pointer may be implemented in Nickolls's thread-based shared memory model. In fact, the only attempt to bridge the models comes from Petitioner's expert, Dr. Rosing, who states (while referring to addresses in Nickolls which are computed based on thread IDs): "A POSITA would have understood that this [computed] address could be stored in a variable such as a pointer." Ex. 1003 [Rosing-Decl.] ¶ 157.

Dr. Rosing, however, does not explain how or why a pointer would be used, or what purpose it would serve in Nickolls, or how a POSITA would run the GPU

Gems simulation without such pointers. As Dr. Khatri explains, replacing the thread ID-based address computation of Nickolls with pointers serves no purpose in Nickolls system:

Nickolls, again, has a fully functioning system that has no use for pointers. Threads may access shared memory and have no need to store an address after it is computed. *See* Ex. 1004 [Nickolls] 8:22-25 (“The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID.”). Once a thread computes the address, it knows where the desired data resides. *Id.*, 20:46-51 (“Instructions to access shared memory or shared global register file space advantageously identify locations to be read and/or written as a function of the thread ID. Where such an instruction is encountered, the thread reads its thread ID and computes the target address from the thread ID using the function specified in the CTA program.”). Storing the computed address in a pointer serves no functional purpose. These are two alternate, incompatible systems of addressing memory. In GPU Gems, the `rdbuffer` pointer is used to direct access to a memory location, while in Nickolls an address computation using thread IDs is used for this purpose. Nickolls has no need to accommodate such pointers and neither Petitioner nor Dr. Rosing explains how GPU Gems’s approach, reliant upon the `rdBuffer` pointer, could or would be implemented in Nickolls without an unnecessary and useless pointer.

Ex. 2030 [Khatri-Decl.] ¶ 78.

Accordingly, Petitioner fails to prove that a POSITA would be motivated to combine the numerical simulations of GPU Gems with Nickolls or that a POSITA would be able to do so with a reasonable expectation of success.

3. Petitioner Fails To Demonstrate A POSITA Would Be Motivated Or Able To Combine Kirk/GPU Gems With A Reasonable Expectation Of Success.

Petitioner's second ground, Kirk/GPU Gems, fares no better than its first. Indeed, it fails for much the same reasons. Again, setting aside the fact that GPU Gems fails to teach the claimed pointer-swapping technique, Petitioner also fails to demonstrate that a POSITA would be motivated to add pointer swapping to Kirk's system. As with Nickolls, adding pointer swapping to Kirk solves no existing problem in Kirk and offers no improvement to its system. *See* Section II.A.3(a). And, here again, Petitioner fails to explain how pointer-swapping could actually be implemented in Kirk's system, much less with a reasonable expectation of success. *See* Section II.A.3(b). Petitioner additionally fails to prove a POSITA would be motivated or able to implement GPU Gems's numerical simulation to Kirk with a reasonable expectation of success. *See* Section II.A.3(c).

a. *Petitioner Fails To Demonstrate That Kirk's System Would Benefit From Pointer Swapping.*

The Kirk/GPU Gems combination mirrors the Nickolls/GPU combination. Here, as there, Petitioner contends a POSITA would be motivated to incorporate

the GPU Gems’s (alleged) pointer swapping into Kirk’s system. Pet., 57-58, Petitioner again contends that pointer swapping is beneficial because it offers an efficient way to reuse outputs across computational cycles by avoiding memory copying. Pet., 67, 77. But, as with Nickolls, Petitioner fails to prove that Kirk performs any memory copying that pointer swapping would avoid.

As an initial matter, Petitioner provides no evidence that Kirk presents any data copying problem that pointer swapping could solve. Petitioner cites Kirk at 4:16-20 which states, “[d]ata produced in a pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 may be written to a buffer in graphics memory to be read from during a subsequent pass.” See Pet., 67, 77 (citing Ex. 1005 [Kirk] 4:16-20). As Dr. Khatri explains, this language does not suggest any copying inefficiency—or any problem that pointer swapping could plausibly address:

Kirk at 4:16-20 merely discloses a multi-pass graphics workflow wherein data is written to a buffer and may be reused across passes.

Nothing here suggests any inefficiency in Kirk’s approach, that copying is even occurring, or that this creates some bottleneck that must be optimized or alleviated, or even that pointers are involved at all. In fact, the word “pointer” does not appear anywhere in Kirk’s disclosure. Kirk, in this regard, merely discloses a conventional approach with no

discernible problems. Nothing herein suggests that pointer swapping would somehow improve Kirk's system.

Ex. 2030 [Khatri-Decl.] ¶¶ 82-83.

In fact, Kirk's disclosure confirms that pointer swapping would neither benefit its system nor be consistent with its core design. As Dr. Khatri explains:

Kirk already avoids the very inefficiency—unnecessary data copying between buffers—that Petitioner alleges would justify pointer swapping. Whereas some prior systems may have required copying to manage parallel processing of fragments, or thread execution, Kirk achieves that goal through real-time conflict detection.

Specifically, Kirk is directed to solving the problem of read-after-write (RAW) conflicts in fragment processing pipelines—*i.e.*, the need to read from a buffer that has recently been written to without flushing the pipeline. *See* Ex. 1005 [Kirk] 1:34–36. Kirk addresses this by introducing a Conflict Detection Unit (CDU), which identifies RAW position conflicts and temporarily locks processing of fragments that would read from memory locations not yet safe to access. *See id.*, 7:3–14. This mechanism allows Kirk to coordinate buffer use without any need to flush buffer data or use pointers. Kirk's system, thus, enables automated concurrent read and write access to the same position in a buffer, eliminating the need for, and conceptual applicability of, a pointer-swapping scheme.

Ex. 2030 [Khatri-Decl.] ¶¶ 85-86.

Thus, Kirk's system locks reads from a buffer while writes are pending, without the need for flushing. Petitioner offers no reason why a POSITA would be motivated to add pointer swapping to Kirk's conflict-management system.

b. *Petitioner Fails To Explain How Pointer Swapping Could Be Implemented In Kirk, Much Less With A Reasonable Expectation Of Success.*

Even if pointer swapping could offer some theoretical benefit, Petitioner fails to explain how it could be implemented in Kirk's system or how a POSITA could do so with a reasonable expectation of success. In fact, Kirk's architecture is fundamentally incompatible with pointer swapping.

As Dr. Khatri explains, Kirk's Conflict Detection Unit (CDU) relies on tracking fixed buffer positions. Introducing pointer swapping would undermine that mechanism and render it inoperative:

Again, Kirk addresses read-after-write (RAW) hazards through a Conflict Detection Unit (CDU) that tracks specific buffer positions. *See* Ex. 1005 [Kirk] 7:8–11, 7:30–35, 8:21–36. The CDU prevents reads from a given buffer position until the corresponding write completes and is confirmed. *Id.*, 7:8–29. Crucially, the CDU's logic depends on fixed, position-specific operation: the CDU system must know the buffer position from where data will be read and written to, in order to detect and resolve conflicts. *See id.*

If Kirk's system were to swap pointers of buffer positions that the CDU is relying on, the CDU's mapping of tracked buffer positions would be corrupted by the pointer swap, and thus would be invalidated, disrupting conflict detection entirely. A consistent operation of Kirk's CDU thus requires stable, unchanging buffer positions to function properly. Implementing pointer swapping by reassigning inputs and outputs (which would reassign buffer positions) would break the link between the CDU's position-based tracking and actual memory writes, disrupting the write-confirmation cycle at the heart of Kirk's RAW hazard mitigation.

Ex. 2030 [Khatri-Decl.] ¶¶ 89-90. These are not superficial implementation issues, but fundamental issues that compromise correct operation. Per Dr. Khatri:

Kirk's conflict detection logic is integral to his system and present within every embodiment. Ex. 1005 [Kirk] cl. 1, cl. 8. Retrofitting it to accommodate pointer-swapped memory would require wholesale architectural redesign. Petitioner provides no analysis of how such a redesign could be implemented and how correctness would continue to be assured and offers no basis for believing a POSITA could achieve it with any reasonable expectation of success. Nor does Dr. Rosing offer any implementation analysis specific to Kirk's system. Her only claim is that pointer swapping would be a "typical method" for enabling data reuse in Kirk, citing Kirk's statement that data may be written to a buffer and later read. *See* Ex. 1003 [Rosing-Decl.] ¶ 309 (citing Ex. 1005, 4:16–20). But that passage refers to conventional buffer reuse, not buffer position tracking for RAW hazard mitigation,

and her testimony includes no analysis of how pointer swapping would interact with Kirk’s CDU, memory address logic, or RAW conflict tracking. The remainder of her discussion is generic and not directed to the architecture of Kirk’s system. *See* Ex. 1003, ¶ 311.

Ex. 2030 [Khatri-Decl.] ¶ 92. In this regard, Dr. Rosing provides no technical description, no integration model, and no memory mapping explanation—just a conclusory assertion that the combination could be easily implemented. Such *ipse dixit*, however, is entitled to little weight. *See* 37 C.F.R. § 42.65(a); *see also* *Xerox*, IPR2022-00624, Paper 9 at 15–16 (Aug. 24, 2022) (precedential) (“conclusory and unsupported [expert testimony] adds little to the conclusory assertion for which it is offered to support”).

In sum, Petitioner has failed to demonstrate that pointer swapping is even implementable in Kirk, let alone that a POSITA would have a reasonable expectation of success in doing so. The proposed modification is incompatible with Kirk’s buffer position-tracking architecture and Petitioner offers no explanation as to how this incompatibility could be addressed or resolved.

c. *Petitioner Fails To Demonstrate A POSITA Would Be Motivated Or Able To Add GPU Gems’s “Numerical Simulation” To Kirk’s System With A Reasonable Expectation Of Success.*

Although Petitioner’s theory is centered on pointer swapping, it relatedly contends that a POSITA would have been motivated to implement GPU Gems’s

numerical simulation in Kirk’s architecture, citing GPU Gems’s teaching that pointer swapping provides benefits “while performing numerical computations,” such as “fast and efficient computation of complex numerical simulations.”⁵ Pet. 57–58 (citing Ex. 1003 ¶¶ 277–79). Petitioner asserts that Kirk’s system, with its parallel execution pipelines, would have been suitable for implementing these techniques, and that a POSITA “would have had a reasonable expectation of success” in doing so. *Id.* But Petitioner’s argument is conclusory, misreads both GPU Gems and Kirk, and fails to grapple with the fundamental architectural and operational differences between them.

⁵ Petitioner’s contention that a POSITA would be motivated to implement GPU Gems’s numerical simulations into Kirk is directly tied to its errant arguments concerning pointer swapping. *See* Pet., 57 (“a POSITA understanding the benefits of swapping pointers while performing numerical computations (as taught in GPU Gems) would have been motivated to apply those techniques when writing programs to perform numerical computations on the accelerator of Kirk.”). Consequently, Petitioner’s arguments concerning implementing numerical computations in Kirk fail for the additional reasons set forth in Sections II.A.3(a)-(b).

Critically, Petitioner’s combination ignores that Kirk’s focus is improving graphics processing. Kirk is directed towards a highly specific graphics-processing use case. Its “field of the invention” explains that it “generally relate[s] to *processing graphics data* in a programmable graphics processor, and more particularly to *writing and reading graphics data* associated with a buffer.” Ex. 1005 [Kirk] 1:7-10. Kirk’s background of the invention describes the problem: when shading texture data is written to a buffer, subsequent reads from that buffer must wait for the pipeline to flush, in order to avoid read-after-write (RAW) conflicts. *Id.*, 1:21–36. Kirk proposes a solution: using a Conflict Detection Unit (CDU) that tracks buffer position accesses and locks conflicting positions without requiring a pipeline flush. *See, e.g., id.*, Abstract; 7:8-35, 8:21-36. Kirk’s invention is thus focused on enabling buffer position usability, without flushing the entire buffer, within a graphics fragment shading context, a goal that has no relationship to general-purpose or numerical computation.

Petitioner’s theory that a POSITA would be motivated to implement GPU Gems’s numerical simulation in Kirk’s graphics shading architecture is not grounded in any disclosure in GPU Gems or Kirk. It is driven by hindsight. As the Federal Circuit has held, “obviousness requires the additional showing that a person of ordinary skill at the time of the invention would have selected and

combined [the] prior art elements in the normal course of research and development to yield the claimed invention.” *Unigene Labs., Inc. v. Apotex, Inc.*, 655 F.3d 1352, 1360 (Fed. Cir. 2011); *In re Ratti*, 270 F.2d 810, 813 (CCPA 1959) (proffered combination is not proper where it would change the basic principles of operation of one of the references).

Petitioner offers no explanation for why a POSITA seeking to implement the simulation described in GPU Gems Chapter 31 would have looked to Kirk’s system. GPU Gems describes a chemical reaction-diffusion simulation implemented using a stream programming model, designed for general-purpose, data-parallel computation. *See* Ex. 1006, 493-494 (chapter “appl[ies] the stream processing concepts introduced in Chapter 29 ... to general-purpose computation on GPUs (GPGPU).”). That simulation has no connection to Kirk’s focus, which is addressing a specific read after write (RAW) hazard in fragment-level graphics rendering.

Petitioner further has not demonstrated that GPU Gem’s simulation could be implemented in or performed in the system of Kirk. Petitioner’s assertion that a POSITA “would have understood” how to implement the simulation on Kirk’s hardware (Pet. 57–58) is unaccompanied by any technical explanation, control logic, or memory mapping to support that claim. *See Netflix, Inc. v. DivX, LLC*, 84

F.4th 1371, 1377 (Fed. Cir. 2023) (“[I]t is the petitioner’s burden to present a clear argument.”). Without any architecture-specific showing, Petitioner’s position is unsupported and entitled to little weight.

Petitioner therefore fails to demonstrate that a POSITA would have been motivated to combine GPU Gems with Kirk or had a reasonable expectation of success in doing so.

B. Petitioner Fails To Show The Combinations Satisfy Numerous Limitations (All Claims, All Grounds).

As discussed, Petitioner fails to demonstrate that a POSITA would be motivated or able to make either of its combinations with a reasonable expectation of success. *See* Section II.A, *supra*. But even if the combinations would and could be made, Petitioner still fails to show that the resulting limitations satisfy numerous limitations.

1. Petitioner Fails To Show That GPU Gems Discloses Or Renders Obvious Swapping Pointers (Claims 1-8, 16-19, All Grounds).

Claims 1-8 and 16-19 require “swap[ping]” the “first pointer” and the “second pointer.” *See* Ex. 1001 [’867] cls. 1[e][iv], 16[e]. As the ’867 patent explains, swapping pointers in this fashion avoids the need to “[c]opy[] textures” which is “an expensive operation computationally” but “since the textures are referred to by texture IDs (pointers), swapping these pointers for input and output

textures after each time step achieves the same result at a much lesser cost.” Ex.

1001 [’867] 11:38-42; *see also id.*, 6:16-18.

Petitioner concedes that neither Nickolls nor Kirk explicitly discloses “pointer swapping.” *See* Pet., 12 (“Nickolls does not explicitly provide the well-known technique of pointer swapping”), 33, 52 (“Kirk does not explicitly state that it uses the well-known technique of pointer swapping”); *see also* Ex. 1003 [Rosing-Decl.] ¶¶ 171, 309. Petitioner does not contend that Nickolls or Kirk implicitly or inherently discloses (or renders obvious) “pointer swapping.”

Instead, as discussed in Section II.A.1(a)-(b), Petitioner contends that pointer swapping is disclosed by GPU Gems:

While Nickolls does not explicitly provide the well-known technique of pointer swapping as an example of how to do this, ***GPU Gems does. GPU Gems describes and provides code*** for a numerical simulation of a chemical reaction-diffusion on a GPU (such as that in Nickolls) ***that swaps pointers to input and output data as claimed.*** Thus, the combination of Nickolls and GPU Gems renders obvious all Challenged Claims.

Pet., 12-13; *see also, id.*, 17 (“GPU Gems describes the concept of swapping pointers of input and output data, as claimed in the ’867 patent.”); 33 (“swapping pointers to input and output data ... is disclosed in GPU Gems.”), 45, 52 (“While Kirk does not explicitly state that it uses the well-known technique of pointer

swapping, GPU Gems does.”).

Specifically, Petitioner contends that the line of code, `std::swap`
(`currentSource`, `currentTarget`), in Listing 31-2 discloses swapping
the “first pointer” (allegedly, `currentSource`) with the “second pointer”
(allegedly, `currentTarget`):

Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

Pet., 34 (relying upon Ex. 1006 [GPU-Gems] 508); *see also, id.*, 18, 52, 67, 77.

But, as Section II.A.1(b) demonstrates, Petitioner fails to demonstrate that
either `currentSource` or `currentTarget` is a “pointer.” Petitioner’s
analysis is wholly conclusory. In fact, as more fully explained in Section II.A.1(c),
incorporated by reference, there is no reason to believe that either
`currentSource` or `currentTarget` is a pointer and ample reason to believe
that they are not. GPU Gems’s code is written in C++, and in that language, a
pointer is typically declared through an asterisk (*). But, as a review of GPU

Gems's source code confirms, neither `currentSource` nor `currentTarget` is declared as a pointer with an asterisk or through any other accepted syntax. *See* Section II.A.1(c), *see also* Ex. 2030 [Khatri-Decl.] ¶¶ 98-100. In fact, they are an enumerated type (declared on lines 27-32 of the file `pug.h`, which is included on line 9 of Ex. 2031 [`gpgpu_reactiondiffusion.cpp`]). *See* Ex. 2038 [`pug.h`]. *Id.*

Consequently, Petitioner fails to carry its burden of demonstrating that either Nickolls/GPU Gems or Kirk/GPU Gems discloses or renders obvious the pointer swapping limitations of claims 1-8 and 16-19.

2. Petitioner's Inconsistent Identification Of The "First Pointer" Violates The Antecedent Basis Rule (Claims 16-19).

Even setting aside the fatal flaw that GPU Gems does not disclose pointer swapping, the Petition fails for an independent reason: Petitioner's pointer mapping violates the antecedent basis rule. Claim 16, again, requires a "first pointer" that is swapped with a "second pointer." As discussed below, however, Petitioner inconsistently maps "the first pointer" to `rdBuffer` in limitation 16[b], but later relies on `currentSource` (or, perhaps, `currentTarget`) as "the first pointer" in limitation 16[e]. In either case, it no longer relies upon `rdBuffer`. That shift in mapping violates the antecedent basis rule.

Claim limitation 16[b] introduces a "first pointer." Limitation 16[e] refers back to that "first pointer" and requires "swapping *the* first pointer with the second

pointer ...” Under the antecedent basis rule, limitation 16[e] requires that the very same “first pointer” previously identified in limitation 16[b] be swapped. *Wi-LAN, Inc. v. Apple Inc.*, 811 F.3d 455, 462 (Fed. Cir. 2016) (“Subsequent use of the definite articles ‘the’ or ‘said’ in a claim refers back to the same term recited earlier in the claim.”) (citation omitted).

Petitioner, however, fails to consistently identify the same “first pointer.” For limitation 16[b], Petitioner relies upon GPU Gems’s “rdBuffer” as the first pointer. Pet., 28, *see also id.*, 63 (incorporating Ground 1 argument). Petitioner cites the following line of code to show how GPU Gems teaches “transferring first input data into a first partition, referenced by first pointer” as required by 16[b] and specifically identifies “rdBuffer” as the pointer:

Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

Id. (citing Ex. 1006 [GPU-Gems] 504, 507); *see also id.*, (“the function

`pgInitBuffer ()` transfers the input data in array to a buffer (partition) that is referenced by a *pointer (i.e., **rdBuffer**)*; *see also id.*, 63 (“GPU Gems teaches this limitation, as explained in Ground 1.”).

Thus, Petitioner clearly identifies `rdBuffer` as the “first pointer.” And, notably, Petitioner had no choice but to identify `rdBuffer` as the “first pointer,” because limitation 16[b] requires transferring input data “into a first partition, referenced by [the] first pointer.” The only element in Listing 31-2 of GPU Gems that could even arguably be a pointer that references a partition where input data is transferred is `rdBuffer`, as there is no other transfer of input data in Listing 31-2’s numerical simulation. *See* Ex. 1006 [GPU-Gems] 507-508.

However, when it comes to limitation 16[e] which requires swapping the “first pointer” with the “second pointer,” Petitioner abandons “`rdBuffer`” and switches to seemingly relying upon “`currentSource`” (or at least `currentTarget`) as the “first pointer.” Petitioner cites the following line of code to support its mapping under limitation 16[e]:

Listing 31-2 (continued). C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
// Initialize the state of the simulation with values in array
pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
    pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
    PUGRect range(0, width, 0, height);
    pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

    std::swap(currentSource, currentTarget);
}
```

Pet., 33-34; *see also* Pet., 67 (“GPU Gems teaches that this would be accomplished ... by swapping the first pointer with the second pointer. Section IV.A.3.a.vi”, citing Ground 1).

Petitioner explains that “[a]t the end of each iteration, the code calls the “swap () ’ function, which swaps the pointers for the `currentSource` buffer with that of the `currentTarget` buffer, thus allowing the output of one computational cycle to become the input for the next computational cycle.”⁶ *Id.*

⁶ In limitation 16[e], Petitioner does not expressly state which of `currentSource` and `currentTarget` is allegedly the “first pointer” and

Neither `currentSource` nor `currentTarget` is `rdBuffer`, the earlier identified first pointer Petitioner uses for limitation 16[b]. In fact, Petitioner doesn't even mention `rdBuffer` in its discussion of 16[e]. Pet., 33-35; *see also id.*, 67 (cross-referencing Ground 1 argument). Petitioner does not address this discrepancy at all.

Petitioner's analysis shifts for convenience. To satisfy 16[b], which requires a pointer referencing a partition receiving input data, Petitioner relies on `rdBuffer`, the only plausible candidate. But when faced with 16[e]'s requirement that the same first pointer be swapped, Petitioner abandons `rdBuffer` and instead points to the only visible "swap" in the code, between `currentSource` and `currentTarget`. That swap does not involve `rdBuffer`. Petitioner's opportunistic remapping violates the claim's structure

which is the "second pointer." *See* Pet., 33-35. In the immediately preceding limitation, 16[d], Petitioner seems to rely upon `currentTarget` as the "second pointer." *See* Pet., 31-33. Consequently, read in context, this may suggest that Petitioner relies upon `currentSource` as the "first pointer" in limitation 16[e]. *See id.*, 33-35. Either way, neither `currentSource` nor `currentTarget` is the first pointer identified in limitation 16[b] (*i.e.*, `rdBuffer`). *See* Pet., 28.

and fails to satisfy limitation 16[e]. *Whatfix, Inc. v. WalkMe Ltd.*, IPR2024-00065, Paper 34, 37-40 (Mar. 28, 2025) (Petitioner’s inconsistent identification of what GUI it was relying upon violated the antecedent basis rule); *Phison Elecs. Corp. v. Vervain, LLC*, PGR2024-00048, Paper 9, 45-47 (Apr. 28, 2025) (Petitioner’s “unexplained inconsistency” violated the antecedent basis rule).

This is not a minor inconsistency. The pointer-swapping mechanism is central to the claimed invention’s performance advantages. Petitioner’s shifting mapping undercuts its own theory of obviousness. The Board should not overlook this inconsistency; it must require Petitioner to show that the “first” and “second” pointers it originally identified are the ones being swapped. Petitioner has not done so.

3. Petitioner’s Mapping Of “The First Pointer” And “The Second Pointer” Is Unclear And Fails Under Either Colorable Interpretation (Claims 1-8).

Petitioner’s approach to Claims 1–8 suffers from the same foundational flaw as its analysis of Claims 16–19: a failure to clearly and consistently map the “first pointer” and “second pointer” required by the claims. This failure is not merely semantic—it renders Petitioner’s mapping of the pointer-swapping limitation (1[e][iv]) internally incoherent and legally insufficient.

Petitioner’s identification of the claimed pointers is ambiguous and contradictory. In claim 1:

- For limitation 1[d][ii][1], Petitioner appears to identify `currentTarget` as the “first pointer.”
- For limitation 1[d][ii][2], Petitioner cites 16[b] (which maps `rdBuffer` as the *first pointer*) and 16[d] (which maps `currentTarget` as the second pointer)—leaving it unclear whether the *second pointer* is `rdBuffer`, `currentTarget`, or both.
- For limitation 1[e][iv], which requires swapping “the first pointer” and “the second pointer,” Petitioner relies on the “`std::swap(currentSource, currentTarget;”` line in GPU Gems.
- From the above, “first pointer” appears to be mapped to `currentTarget`, while “second pointer” appears to be mapped to `rdbuffer` or `currentTarget`. But the code cited by Petitioner for limitation 1[e][iv] swaps neither `rdBuffer` with `currentTarget` nor `currentTarget` with itself—and thus fails to demonstrate the required swap under any colorable mapping.

As with claim 16, Petitioner offers no coherent explanation for how this swap satisfies limitation 1[e][iv] in light of its prior pointer identifications.

In limitation 1[d][ii][1], the “first pointer” is introduced as referencing a partition that stores first output data. Petitioner refers to its analysis of limitation 16[d] and argues that what was called the “second pointer” in claim 16 becomes the “first pointer” in claim 1. *See* Pet., 43, 74. In 16[d], Petitioner identified `currentTarget` as the second pointer (Pet., 32–33), so it now appears to identify `currentTarget` as the first pointer in claim 1.

For limitation 1[d][ii][2], which introduces the “second pointer,” Petitioner again refers back to 16[b] and 16[d]. Pet., 43, 75. But those references are inconsistent—16[b] maps the first pointer to `rdBuffer`, while 16[d] maps the second pointer to `currentTarget`. Thus, under one reading, Petitioner is mapping `currentTarget` as both the first and second pointer in claim 1.

That, however, defies common sense and basic claim construction principles. “Where a claim lists elements separately, ‘the clear implication of the claim language’ is that those elements are ‘distinct component[s]’ of the patented invention.” *Becton, Dickinson & Co. v. Tyco Healthcare Group, LP*, 616 F.3d 1249, 1254 (Fed. Cir. 2010) (citation omitted). Likewise, when a claim uses numbered terms like “first” and “second,” those terms refer to separate and distinct

instances of the element. *see Gillette Co. v. Energizer Holdings, Inc.*, 405 F.3d 1367, 1373 (Fed. Cir. 2005) (The terms “first,” “second,” and “third” are terms used to distinguish separate elements of a claim.) *see also 3M Innovative Props. Co. v. Avery Dennison Corp.*, 350 F.3d 1365, 1371 (Fed. Cir. 2005) (“The use of the terms ‘first’ and ‘second’ is a common patent-law convention to distinguish between repeated instances of an element or limitation.”). Under these precedents, `currentTarget` cannot be both the “first pointer” and “second pointer” in the same claim.

But even if that inconsistency were overlooked, Petitioner still cannot satisfy the pointer-swapping requirement of limitation 1[e][iv]. The cited code—`“std::swap(currentSource, currentTarget)”`—does not swap `currentTarget` with itself and does not involve `rdBuffer` at all. *See Pet.*, 45, 77; Ex. 1006 [GPU Gems] 508. Thus, whether the first pointer is `rdBuffer` or `currentTarget`, the code Petitioner relies upon fails to demonstrate a swap between the “first pointer” and “second pointer” as required.

Petitioner’s lack of clarity is itself a fatal defect under 35 U.S.C. § 312(a)(3), which requires that the petition identify each claim element and supporting evidence with particularity. The Board and Federal Circuit have repeatedly emphasized that this includes specific and consistent mapping of claim elements.

See Netflix, 84 F.4th at 1377 (“Ultimately, it is the petitioner's burden to present a clear argument.”); *id.*, 1380 (“[W]e emphasize that it is the petitioner’s burden to make clear when alternative arguments are being presented and to sufficiently expound on each one.”); *Corephotonics, Ltd. v. Apple Inc.*, 84 F.4th 990, 1001 (Fed. Cir. 2023) (“The IPR petition, thus, must provide an understandable explanation of the element-by-element specifics of the patentability challenges, including the identification of particular portions of prior art on which the petitioner is relying.”) (citations omitted).

Here, Petitioner fails to do so. It fails to clearly identify what it believes the first and second pointers are, provides no consistent mapping across limitations, and offers no explanation for how its cited code performs the claimed swap. Accordingly, Petitioner’s argument that GPU Gems discloses “swap[ping] the first pointer and the second pointer” fails.

4. Petitioner Fails To Show That Any Ground Discloses Or Renders Obvious Claim 9.

Petitioner argues that both Nickolls/GPU Gems and Kirk/GPU Gems render claim 9 obvious. But Petitioner’s analysis is cursory and relies almost entirely on citations to limitations in other claims—primarily claims 1 and 16—without addressing the material differences in language and structure. Petitioner does not

meaningfully engage with the actual limitations of claim 9 and further fails to demonstrate that any of the stream limitations are satisfied.

a. *Petitioner's Analysis Fails To Account For The Material Differences In Claim 9.*

Petitioner contends that both Nickolls/GPU Gems and Kirk/GPU Gems render claim 9 obvious, but Petitioner devotes less than two pages to its analysis of claim 9 under each ground. Pet., 49-50, 81-82. For every limitation, Petitioner provides unexplained citations to its analysis under claims 1 and 16:

<p>m. Claim 9</p> <p>i. 9[pre]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶240–41. <i>See 1[pre], 16[pre].</i></p> <p>ii. 9[a]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶242–43. <i>See 1[d][i], 16[c].</i></p> <p>iii. 9[b]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶244–45. <i>See 1[d][ii], 16[b].</i></p> <p>iv. 9[c]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶246–47. <i>See 1[d][i], 1[e], 16[c], 16[d].</i></p> <p>v. 9[c][i]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶248–51. <i>See 1[d][i], 16[c].</i> The computations performed by the processing engines are controlled by a “computational stream” because the processing occurs as part of a sequential execution of a “stream of commands.” Ex1004, 6:65–7:4, 14:17–57,</p>	<p>17:4–36, 20:61–21:8. The “accelerator controller” (<i>i.e.</i>, core interface 128 alone and with instruction unit 312) passes a stream of instructions to the graphics processing unit (<i>e.g.</i>, core 126 and/or processing engine) to perform computations, which then executes the instructions. <i>Id.</i>; Ex1003, ¶¶249–51.</p> <p>vi. 9[c][ii]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶252–53. <i>See 16[d].</i> The transfer of output data from the GPU to accelerator memory is controlled by an “output stream” because the transfer occurs as part of a stream of commands directing output data into accelerator memory. Ex1004, 6:65–7:4, 14:17–57, 17:4–36, 20:61–21:8. The “accelerator controller” (<i>i.e.</i>, core interface 128 alone and with instruction unit 312) passes a stream of instructions to the graphics processing unit (<i>e.g.</i>, core 126 and/or processing engine), which includes a direction to return an output to a location specified by the controller. <i>Id.</i>; Ex1003, ¶¶253.</p> <p>vii. 9[c][iii]</p> <p>Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶254–55. <i>See 1[d], 16[a], 16[d].</i> The transfer of input data to accelerator memory is controlled by an “input stream” because the transfer occurs as part of a stream of commands directing input data into accelerator memory. Ex1004, 13:16–25, 14:16–26, 17:4–18, 19:26–61.</p>
--	---

Pet., 49-50; *see also id.*, 81-82 (Kirk).

Thus, Petitioner relies heavily, without explanation, upon its analysis of other claims. And yet, there are clear differences in these claims and limitations that Petitioner fails to address. By way of example, claim 9 requires “an accelerator controller, operably coupled to ... the at least one graphics processing unit.” Ex. 1001 [’867] 9[c]. Petitioner’s entire analysis as to this limitation is an improper and unexplained incorporation by reference of its expert’s testimony and an unexplained reference to limitations 1[d][i], 1[e], 16[c], and 16[d]:

iv. 9[c]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶246–47.

See 1[d][i], 1[e], 16[c], 16[d].

Pet., 49; *see also id.*, 81 (same).

As an initial matter, the Board and Patent Owner should not be expected to suss out Petitioner’s argument from its unexplained citations. As the Federal Circuit has explained:

While the Board should not take an overly mechanistic view of a petition and decline to address an argument because the petitioner did not present it with ideal vigor and clarity, the Board should also not have to decode a petition to locate additional arguments beyond the ones clearly made. A petitioner may not rely on a vague, generic, and/or meandering petition and later fault the Board for failing to understand what the petition really meant. Ultimately, it is the

petitioner's burden to present a clear argument. *Intelligent Bio-Systems*, 821 F.3d at 1367 (“It is of the utmost importance that petitioners in the IPR proceedings adhere to the requirement that the initial petition identify ‘with particularity’ the ‘evidence that supports the grounds for the challenge to each claim.’”) (quoting 35 U.S.C. § 312(a)(3) (2012)); *see also* 37 C.F.R. § 42.104(b)(4)-(5) (2012) (requiring a petitioner to identify “[h]ow the construed claim is unpatentable,” “specify where each element of the claim is found in the prior art patents or printed publications relied upon,” and identify “[t]he exhibit number of the supporting evidence relied upon to support the challenge and the relevance of the evidence to the challenge raised, including identifying specific portions of the evidence that support the challenge”).

Netflix, 84 F.4th at 1377; *see also Microsoft Corp v. Biscotti, Inc.*, 878 F.3d 1052, 1074 (Fed. Cir. 2017) (petition’s “bare-bones allegation” insufficient to establish disclosure of limitation); *Apple Inc. v. Contentguard Holdings, Inc.*, IPR2015-00450, Paper 9, 6-7 and 10-11 (June 29, 2015) (rejecting argument based upon “vague, nested citations to ... sections of the Petition” and “conclusory analysis with supporting ... internal cross-references to the Petition.”).

Beyond violating its obligation to set forth its argument with specificity, Petitioner’s invitation to mind read is particularly unwelcome here where **none** of the referenced limitations in claims 1 and 16 recite “an accelerator controller, operably coupled to...the at least one graphics processing unit”:

9[c]: an accelerator controller, operably coupled to the accelerator memory and the at least one graphics processing unit, to execute:

1[d][i]: at least one graphics processing unit to generate second output data, during the second computational cycle, by performing at least one calculation on the first output data

1[e]: an accelerator controller, operably coupled to the accelerator memory and the central processing unit,

16[c]: performing, by at least one graphics processing unit during the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data;

16[d]: storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory; and

Limitation 1[e] gets closest, but it requires “an accelerator controller operably coupled to ... **the central processing unit,**” not the “graphics processing unit.” Nor does Petitioner’s analysis of limitation 1[e] suggest “an accelerator controller, operably coupled to ... the at least one graphics processing unit.” For Nickolls/GPU Gems, Petitioner states: “**Core interface 128** (an ‘accelerator controller’) **is operably coupled to CPU 102** and ‘accelerator memory’ within GPU 122 (e.g., local register file 304, global register file 306, and memory 308).” Pet., 43. Similarly, for Kirk, Petitioner contends that its alleged “accelerator controller” is operably coupled to the “Host Processor 114” (the alleged CPU) not the alleged

GPU. Pet., 75. In both grounds, Petitioner contends the alleged controller is operably coupled to the CPU, but not to the GPU as limitation 9[c] requires.

Likewise, limitation 9[b] requires “an accelerator memory, operably coupled to the at least one graphics processing unit, to store data used to perform the at least one computation.” Ex. 1001 [’867] cl. 9[b]. Here too, Petitioner entirely relies on limitations from claims 1 and 16. *See* Pet., 49 (“Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶244–45. *See* 1[d][ii], 16[b].”); *Id.*, 81 (“Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶376–77. *See* 1[d][ii], 16[b].”). Again, the cited limitations materially differ from limitation 9[b]; neither limitation 1[d][ii] nor 16[b] requires “to store data used to perform the at least one computation.”

This failure to address the actual limitations of claim 9 renders Petitioner’s analysis legally deficient and insufficient to meet its burden under 35 U.S.C. § 312(a)(3). Petitioner has not mapped the actual language of claim 9, has not addressed the structural differences from the claims it relies on, and has not identified how the prior art satisfies the limitations as written. The Petition fails as to claim 9 for this reason alone.

b. *Petitioner Fails to Show That Either Combination Discloses Or Renders Obvious Any Of The Required “Streams.”*

Claim 9 requires “an accelerator controller...to execute” three streams: “a computational stream controlling performance of the at least one computation,” “an output stream controlling transfer of the output data,” and an “input stream controlling transfer of input data.” Ex. 1001 [’867] cl. 9. Petitioner fails to demonstrate that any of the stream limitations are disclosed or rendered obvious by Nickolls/GPU Gems or Kirk/GPU Gems.

Petitioner contends that Nickolls/GPU Gems discloses all three streams. *See* Pet., 49–50. But for both the computational and output streams, Petitioner cites the exact same line of code: a single “stream of commands” written by the CPU to the GPU. Ex. 1004 [Nickolls] 6:65–7:4. Petitioner then appends a string of additional citations (*e.g.*, 14:17–57, 17:4–36, 20:61–21:8) without explaining how those portions relate to any of the claimed streams. No explanation is given for why a single “stream of commands” satisfies either the “computational stream” or the “output stream”—much less both. Nor is any explanation given for why one stream may satisfy two distinct claim elements. *See Becton, Dickinson & Co. v. Tyco Healthcare*, 616 F.3d 1249, 1254 (Fed. Cir. 2010) (“Where a claim lists

elements separately, the clear implication of the claim language is that those elements are distinct”).

Petitioner’s analysis of the input stream fares no better: Petitioner simply asserts that an input stream “direct[s] input data into memory” and again invokes limitations from claims 1 and 16 but fails to identify any “input stream” at all. Pet., 50.

This same pattern repeats itself in Kirk/GPU Gems. There too, Petitioner provides string citations allegedly disclosing each of the “computational stream,” “input stream,” and “output stream.” Pet., 81-82. Petitioner never identifies what, in those citations disclose the three claimed streams.

Petitioner’s argument fails for two reasons. First, it is conclusory. *See* Section II.A.1(b) (citing *KSR*, *Magnum Oil*). Second, it is insufficiently specified. *See* Section II.A.4(a) (citing 35 U.S.C. § 312(a)(3), *Contentguard*; *Netflix*, *Corephotonics*). Petitioner also cites its expert, Dr. Rosing, but her testimony is improperly incorporated, and similarly conclusory. *See* 37 C.F.R. § 42.65(a); *see also Xerox*, IPR2022-00624, Paper 9, 15-16 (Aug. 24, 2022) (precedential) (“conclusory and unsupported [expert testimony] adds little to the conclusory assertion for which it is offered to support, and is entitled to little weight.”).

Accordingly, Petitioner has not shown that any of the three streams required by Claim 9 are disclosed or rendered obvious under either ground.

III. CONCLUSION

For the foregoing reasons, Patent Owner respectfully requests that institution be denied under 35 U.S.C. § 314(a).

Respectfully submitted,

/ Kenneth J. Weatherwax /
Kenneth J. Weatherwax (Reg. No. 54,528)
Nathan Lowenstein, *pro hac vice*
Colette Woo, *pro hac vice*
LOWENSTEIN & WEATHERWAX LLP

Date: July 14, 2025

CERTIFICATE OF COMPLIANCE WITH TYPE-VOLUME LIMITS

This Patent Owner Preliminary Response (the “POPR”) consists of 12,806 words, excluding table of contents, table of authorities, certificate of service, this certificate, or table of exhibits. The POPR complies with the type-volume limitation of 14,000 words as mandated in 37 C.F.R. § 42.24. In preparing this certificate, counsel has relied on the word count of the word-processing system used to prepare the paper (Microsoft Word).

Respectfully submitted,

/ Abbie Neufeld /

Date: July 14, 2025

CERTIFICATE OF SERVICE

The undersigned hereby certifies that the following documents were served
by electronic service, by agreement between the parties, on the date below:

PATENT OWNER PRELIMINARY RESPONSE

EXHIBITS 2030-2039

The names and addresses of the parties being served are as follows:

Brian M. Buroker	bburoker@gibsondunn.com
L. Kieran Kieckhefer	kkieckhefer@gibsondunn.com
Nathan Curtis	ncurtis@gibsondunn.com
Vivian Lu	vlu@gibsondunn.com

Respectfully submitted,

/ Abbie Neufeld /

Date: July 14, 2025