

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

Prior Art Reference: U.S. Patent No. 7,139,003 (“Kirk”)

Kirk was filed on December 15, 2003, and therefore qualifies as prior art to U.S. Patent No. RE48,438 (the “’438 patent”) under at least pre-AIA 35 U.S.C. § 102(e). Kirk anticipates and/or renders obvious claims 1–10, 12–14, 16–18, 20–32, and 40–54 of the ’438 patent (the “Asserted Claims”).

The chart below provides representative examples of where each element of each claim is found within Kirk. Citations are meant to be exemplary, not exhaustive, and NVIDIA reserves the right to identify and discuss additional portions of the reference in support of its contentions and/or to rebut arguments made by Plaintiff. Where NVIDIA states that Kirk “discloses” a limitation, that disclosure may be express, implicit, and/or inherent. Citations to figures, drawings, tables, and the like include reference to any accompanying or related text. All internal cross-references are meant to incorporate the cross-referenced material as if fully set forth therein.

It is NVIDIA’s position that Plaintiff’s Infringement Contentions have not established that any accused product or service infringes any valid claim. Thus, NVIDIA’s statements below should not be treated as an admission, implication, or suggestion that NVIDIA agrees with Plaintiff regarding either the scope, construction, or interpretation of any of the Asserted Claims or the infringement theories advanced by Plaintiff in its Infringement Contentions, including whether any Asserted Claim satisfies 35 U.S.C. §§ 101 or 112. The exemplary disclosures in these invalidity contentions are informed by the apparent interpretation of the asserted claims by Plaintiff. These statements are not intended to suggest that NVIDIA agrees with Plaintiff’s application of any claim term, suggest a proposed construction at this stage of the case, or suggest that construction is needed.

To the extent Plaintiff argues that any element below is not disclosed by Kirk, a person of ordinary skill in the art would have found it obvious to combine the teachings of Kirk with the background knowledge of a person of ordinary skill in the art and/or the additional references, and exemplary teachings, set forth in NVIDIA’s Invalidity Contentions, including all exhibits and appendices thereto. Additional exemplary combinations and reasons for obviousness are described in the cover pleading, and are incorporated by reference herein.

Plaintiff has yet to identify any limitation of the Asserted Claims that it contends is not anticipated and/or rendered obvious by Kirk. NVIDIA therefore expressly reserves the right to respond to any such contention, including by identifying additional obviousness combinations, if Plaintiff makes any such contention. NVIDIA reserves the right to rely on additional citations or sources of evidence that also may be applicable, or that may become applicable in light of claim construction, changes in Plaintiff’s infringement contentions, and/or information obtained during discovery as the case progresses.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>1[pre] “A computer system, comprising:”</p>	<p>To the extent the preamble is limiting, Kirk discloses “[a] computer system.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses a computer system (e.g., computing system 100 and/or graphics processor 105):</i></p> <p>Various embodiments of the invention include an application programming interface for a programmable graphics processor. The application programming interface includes one or more program instruction to configure a fragment processor within the programmable graphics processor to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved. 1:40–47.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem. FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline of FIG. 1A in accordance with one or more aspects of the present invention. 2:14–21.</p> <p>FIG. 12A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem. FIGS. 12B and 12C are block diagrams of exemplary embodiments of the Programmable Graphics Processing Pipeline of FIG. 12A in accordance with one or more aspects of the present invention. 3:1–8.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a Computing System generally designated 100 and including a Host Computer 110 and a Graphics Subsystem 107. Computing System 100 may be a desktop computer, server, laptop computer, palm-sized computer, tablet computer, game console, cellular telephone, computer-based simulator, or the like. Host computer 110 includes Host Processor 114 that may include a system memory controller to interface directly to Host Memory 112 or may communicate with Host Memory 112 through a System Interface 115. System</p>

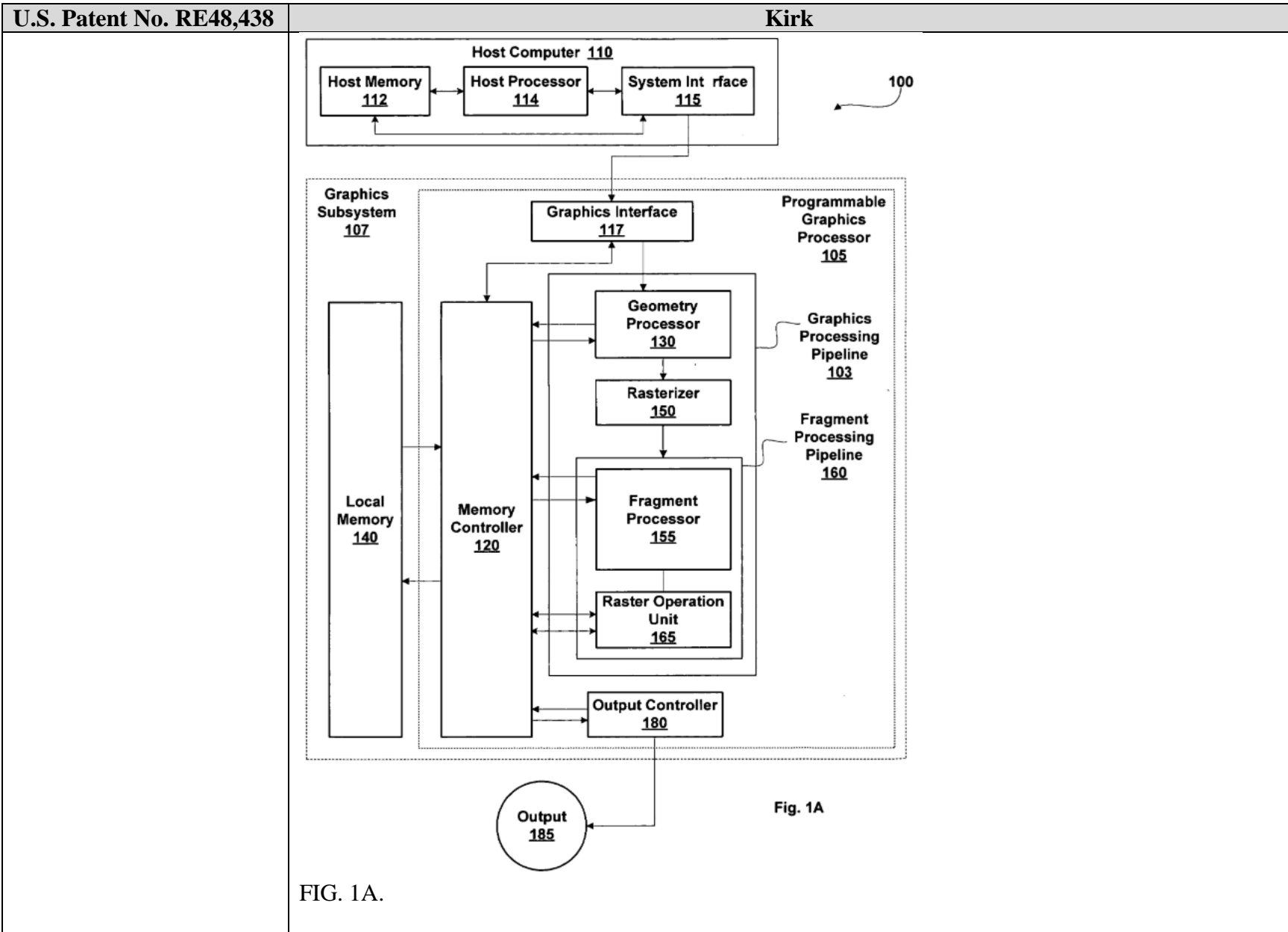
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Interface 115 may be an I/O (input/output) interface or a bridge device including the system memory controller to interface directly to Host Memory 112. Examples of System Interface 115 known in the art include Intel® Northbridge and Intel® Southbridge. 3:16–31.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes,</p>

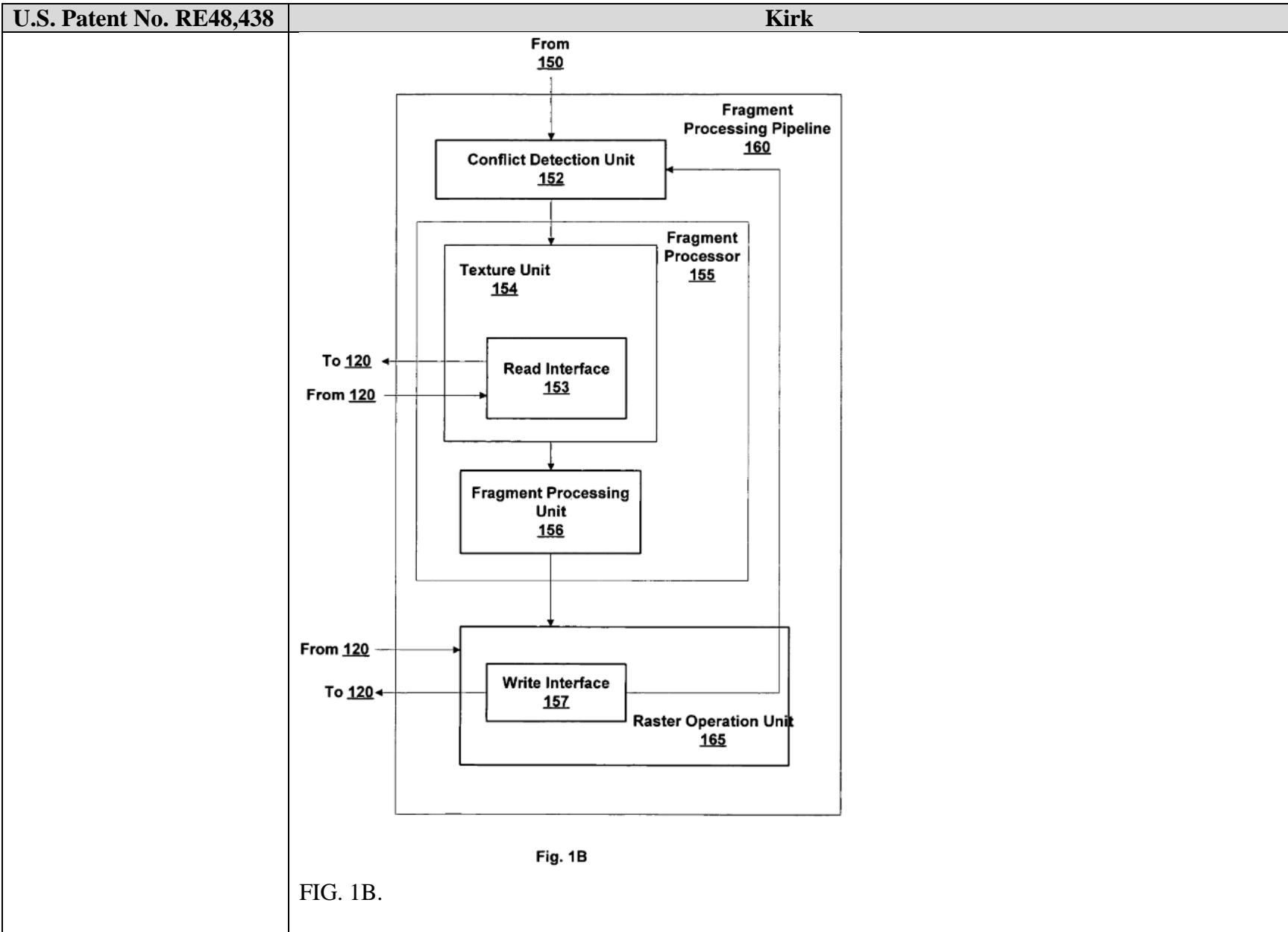
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 26:56–61.</p>

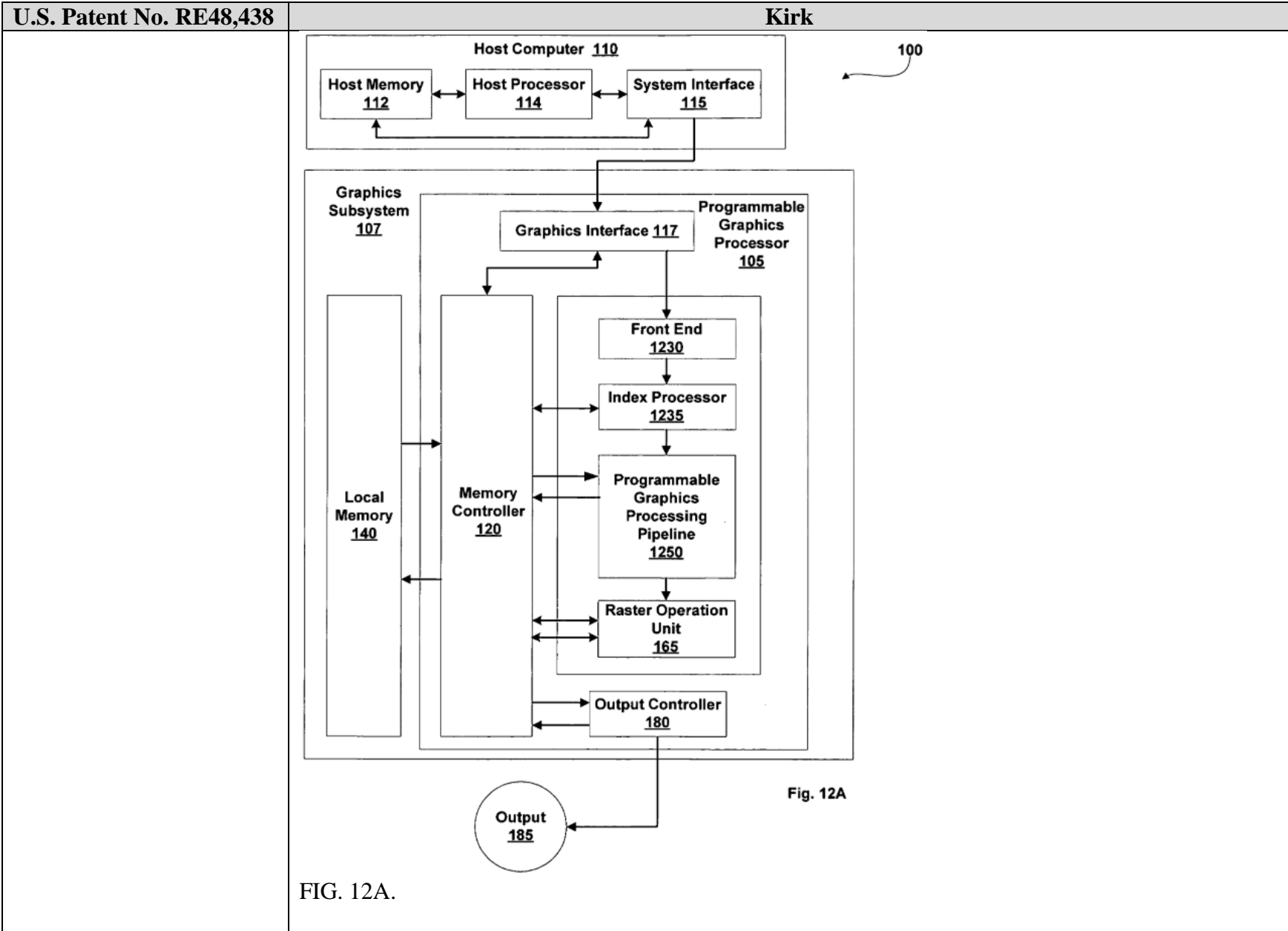
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

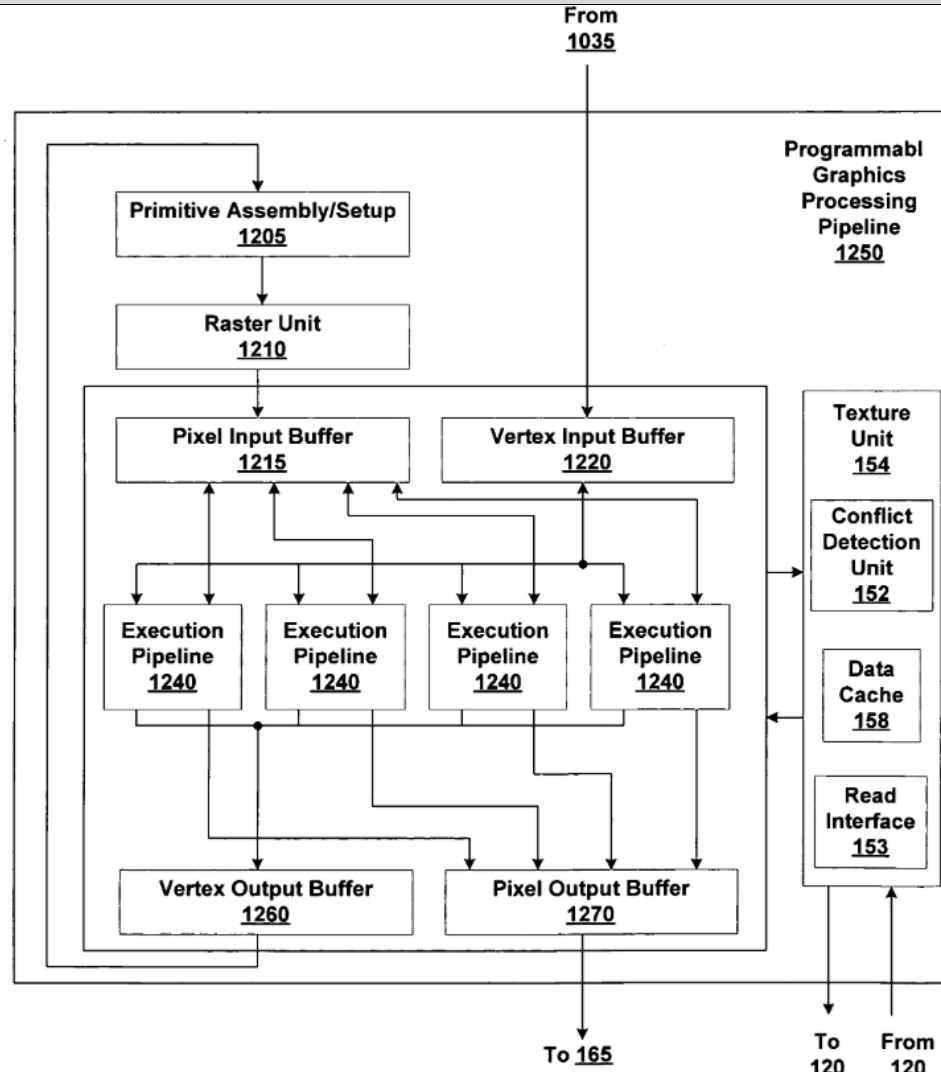


Fig. 12B

FIG. 12B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[a] “a central processing unit to receive input data;”</p>	<p>Kirk discloses “a central processing unit to receive input data.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses a central processing unit (e.g., Host Processor 114) that receives input data (e.g., in order to have it processed by Graphics Subsystem 107):</i></p> <p>Various embodiments of the invention include an application programming interface for a programmable graphics processor. The application programming interface includes one or more program instruction to configure a fragment processor within the programmable graphics processor to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved. 1:40–47.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem. FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline of FIG. 1A in accordance with one or more aspects of the present invention. 2:14–21.</p> <p>FIG. 12A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem. FIGS. 12B and 12C are block diagrams of exemplary embodiments of the Programmable Graphics Processing Pipeline of FIG. 12A in accordance with one or more aspects of the present invention.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>3:1–8.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a Computing System generally designated 100 and including a Host Computer 110 and a Graphics Subsystem 107. Computing System 100 may be a desktop computer, server, laptop computer, palm-sized computer, tablet computer, game console, cellular telephone, computer-based simulator, or the like. Host computer 110 includes Host Processor 114 that may include a system memory controller to interface directly to Host Memory 112 or may communicate with Host Memory 112 through a System Interface 115. System Interface 115 may be an I/O (input/output) interface or a bridge device including the system memory controller to interface directly to Host Memory 112. Examples of System Interface 115 known in the art include Intel® Northbridge and Intel® Southbridge.</p> <p>3:16–31.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like.</p> <p>3:32–46.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are</p>

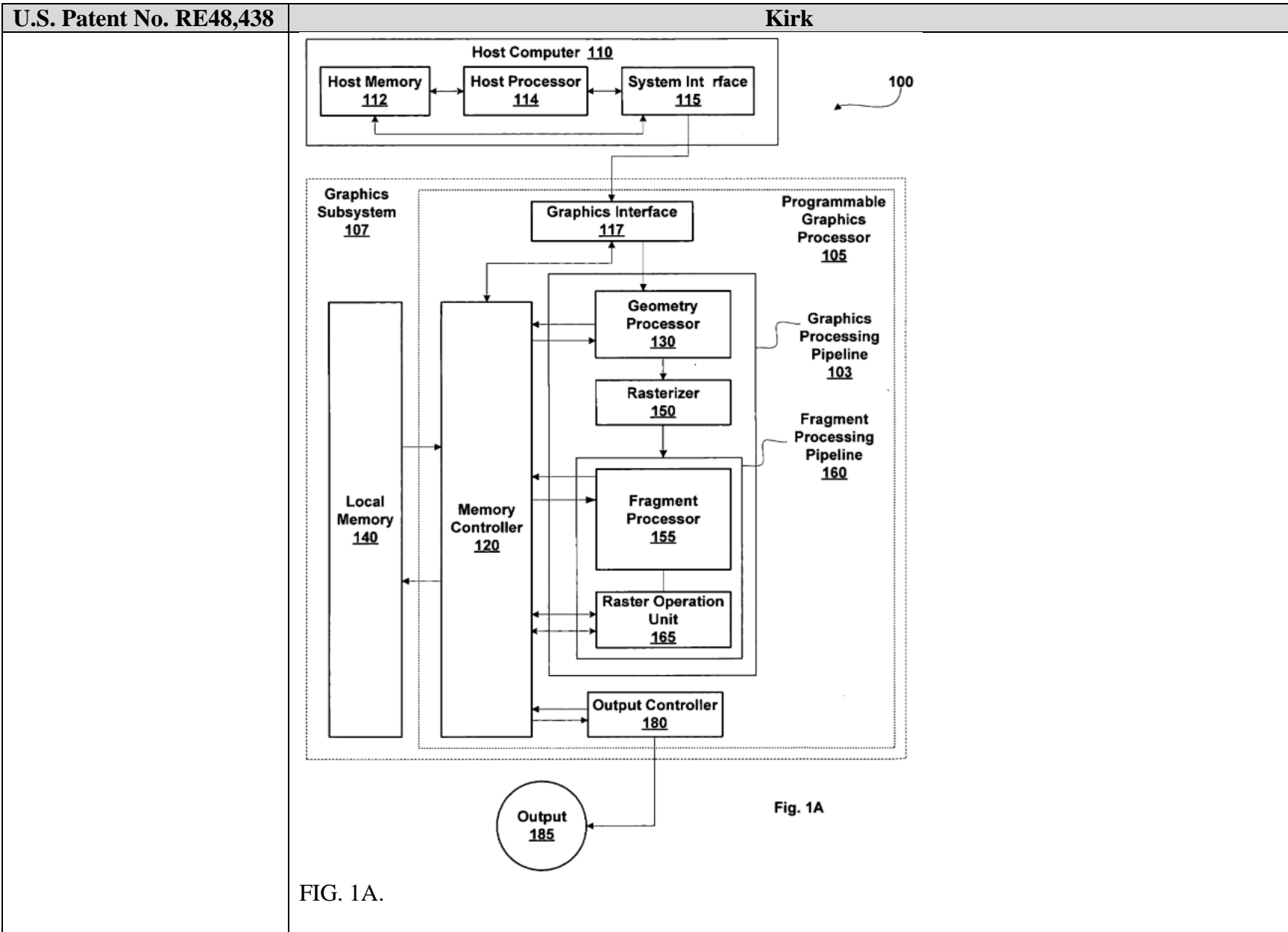
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like.</p>

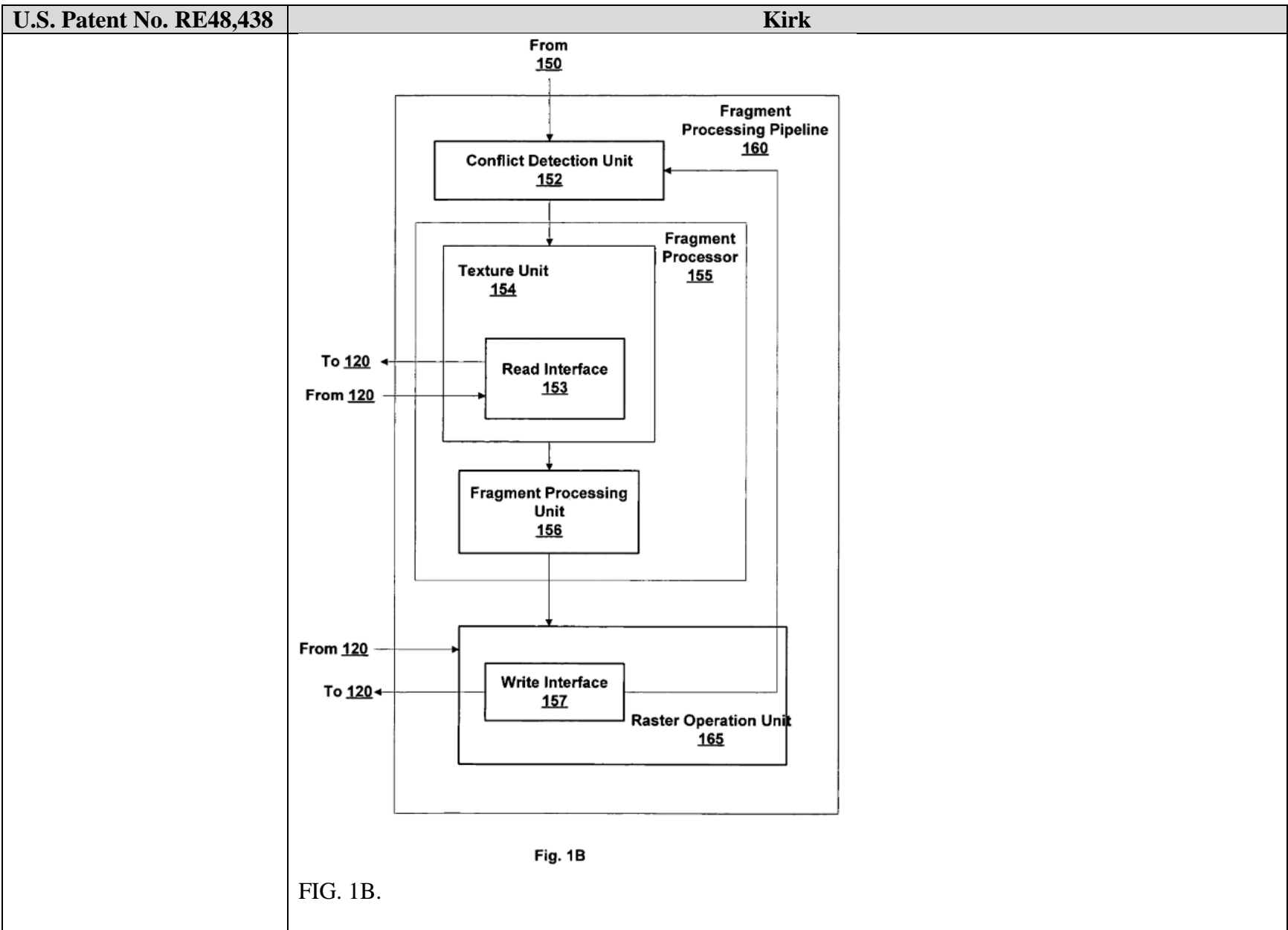
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>6:4–33.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory.</p> <p>26:24–39.</p> <p>In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data.</p> <p>26:56–61.</p>

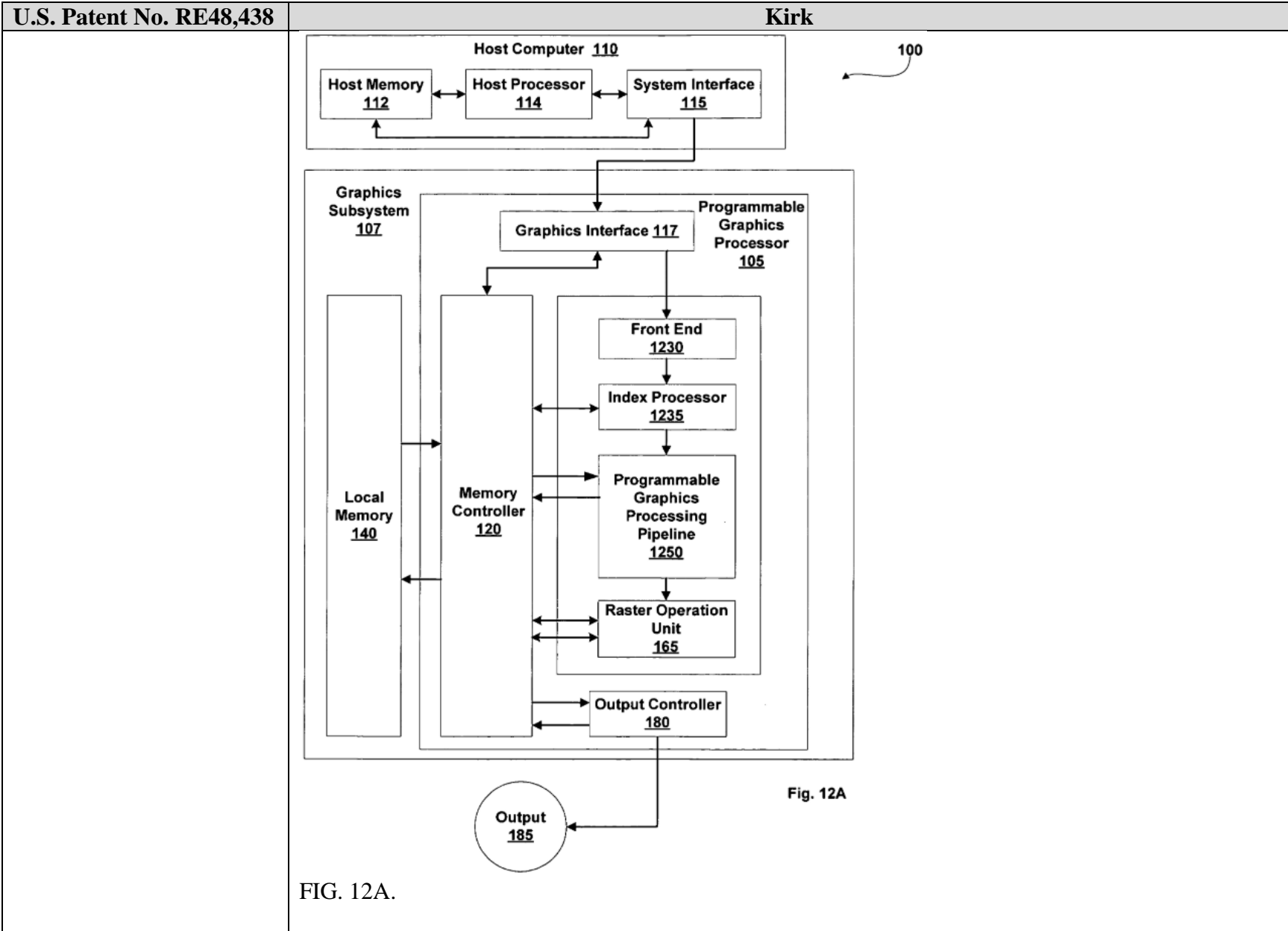
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

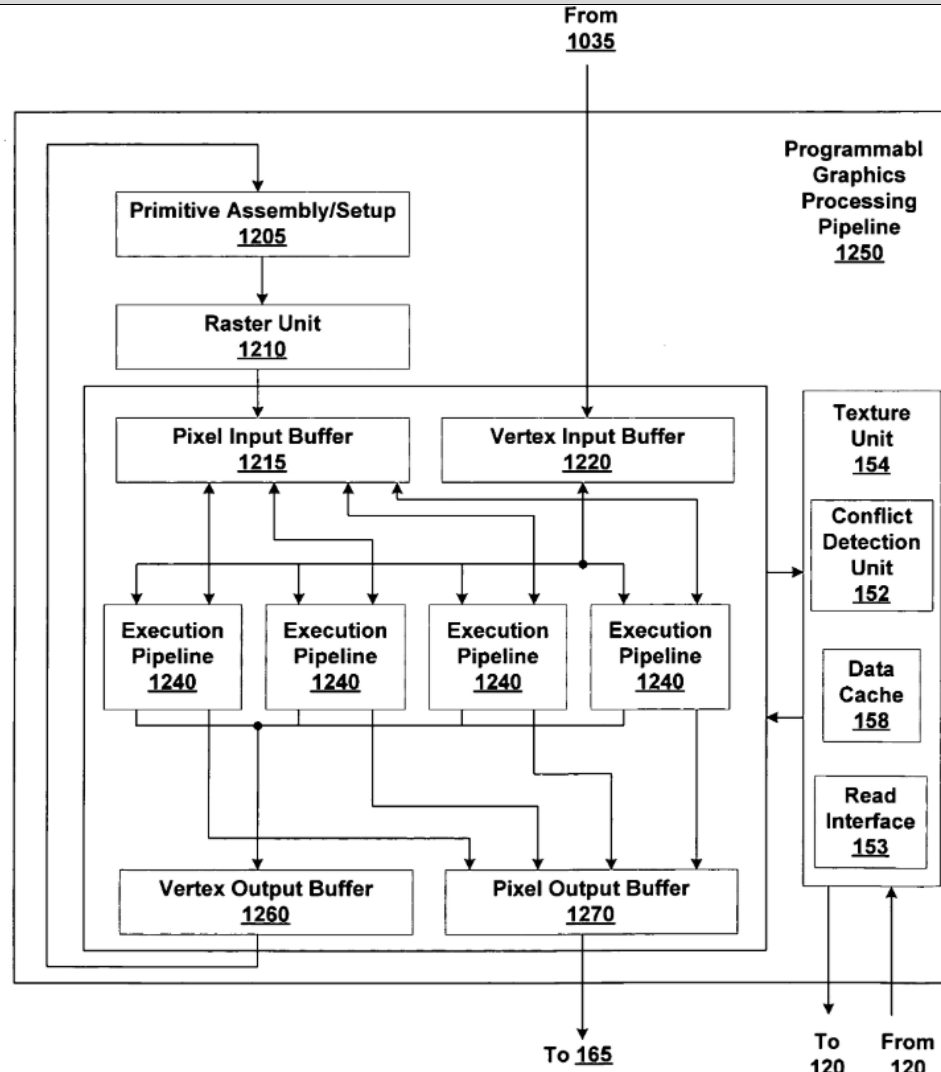


Fig. 12B

FIG. 12B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[b] “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit;”</p>	<p>Kirk discloses “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses main memory (e.g., Host Memory 112), which is coupled to a central processing unit (e.g., Host Processor 114) directly and/or via a bus, and stores input data received by the central processing unit:</i></p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline of FIG. 1A in accordance with one or more aspects of the present invention. 2:14–21.</p> <p>FIG. 12A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem.</p> <p>FIGS. 12B and 12C are block diagrams of exemplary embodiments of the Programmable Graphics Processing Pipeline of FIG. 12A in accordance with one or more aspects of the present invention. 3:1–8.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a Computing System generally designated 100 and including a Host Computer 110 and a Graphics Subsystem 107. Computing System 100 may be a desktop computer, server, laptop computer, palm-sized computer, tablet</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>computer, game console, cellular telephone, computer-based simulator, or the like. Host computer 110 includes Host Processor 114 that may include a system memory controller to interface directly to Host Memory 112 or may communicate with Host Memory 112 through a System Interface 115. System Interface 115 may be an I/O (input/output) interface or a bridge device including the system memory controller to interface directly to Host Memory 112. Examples of System Interface 115 known in the art include Intel® Northbridge and Intel® Southbridge. 3:16–31.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p>

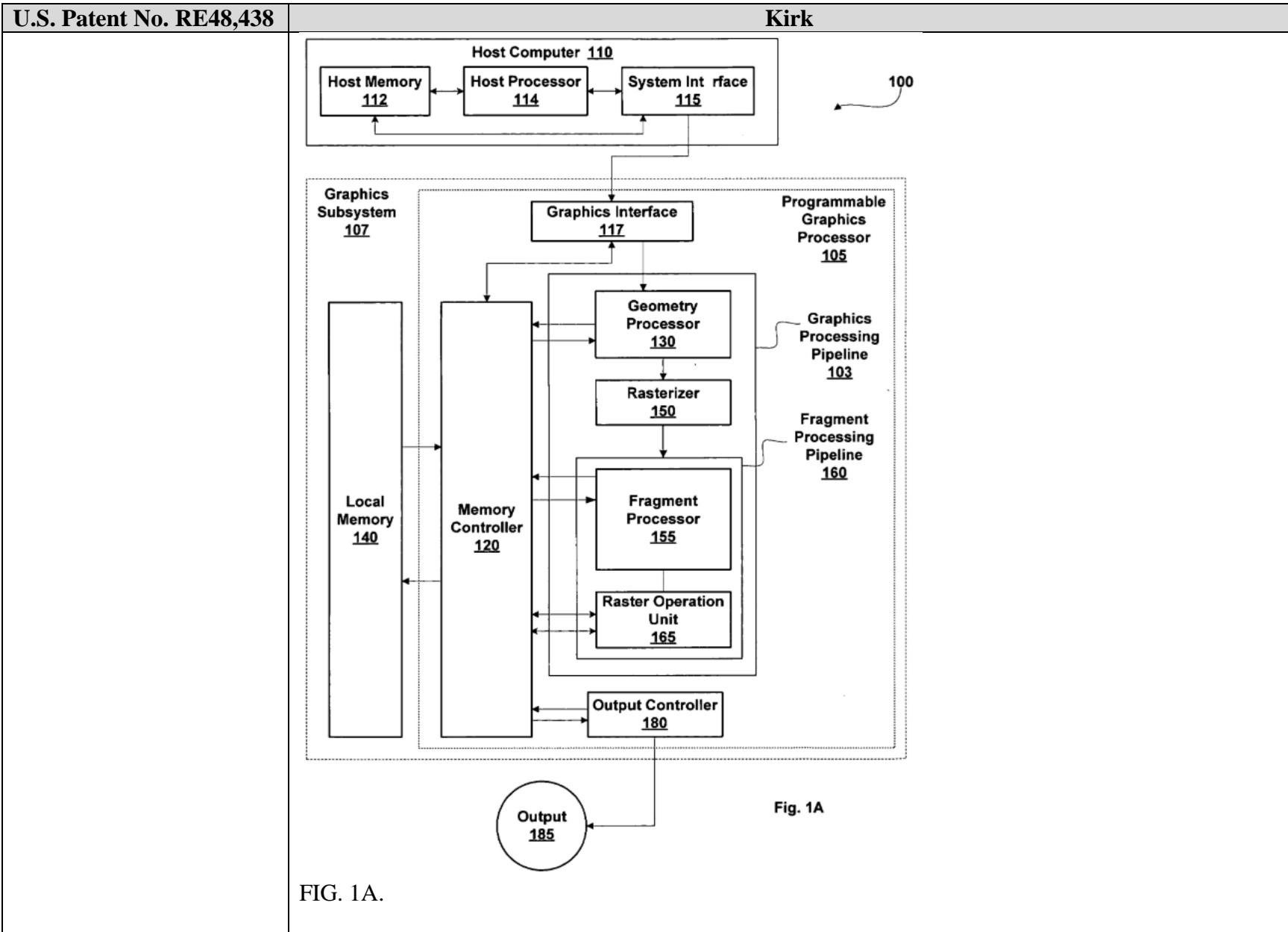
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>In step 209 Conflict Detection Unit 152 determines if a RAW position conflict exists for the position associated with the second fragment, and, if so, in step 211 Conflict Detection Unit 152 locks processing of the second fragment. Locking a fragment prevents any processing of the fragment requiring source data that is not yet available due to a RAW position conflict. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory. Step 214 may be completed several, even hundreds of clock cycles after step 205. Raster Operation Unit 165 outputs the write position information to Fragment Processor 155 confirming that the write is complete. In one embodiment the write is considered complete when the write request is output from Memory Controller 120 to Local Memory 140 or to Host Memory 112 via Graphics Interface 117. In another embodiment the write is considered complete when the write request is output from Raster Operation Unit 165 to Memory Controller 120. In step 217 Fragment Processing Pipeline 160 unlocks the second fragment and proceeds to step 220. In step 220 Fragment Processor 155 begins shading the second fragment as specified by the shader. 7:8–29.</p> <p>FIG. 5 is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160, including a Data Cache 158. Fragment data received by Fragment Processing Pipeline 160 from Rasterizer 150 is processed according to the fragment program instructions and Conflict Detection Unit 152 determines if a RAW conflict exists for each fragment. Conflict Detection Unit 152 outputs fragment data and the fragment program instructions to Fragment Processor 155. A Read Interface 153 within Texture Unit 154 reads additional fragment program instructions and buffer data from Data Cache 158. When the additional fragment program instructions or the buffer data are not available in Data Cache 158, i.e., there is a “cache miss”, the additional fragment program instructions or the buffer data is read from either Local Memory 140 or Host Memory 112, via Memory Controller 120 and optionally stored in Data Cache 158 and output by Read Interface 153. When a location is entered in Conflict Detection Unit 152 for a pending write, Conflict Detection Unit 152 determines if data stored in the location is available in an entry in Data Cache 158 and if so, invalidates the entry in Data Cache 158 as described further herein. In an alternate embodiment, entries in Data Cache 158 containing data read from a location that is entered in Conflict Detection Unit 152 are invalidated by Read Interface 153.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data</p>

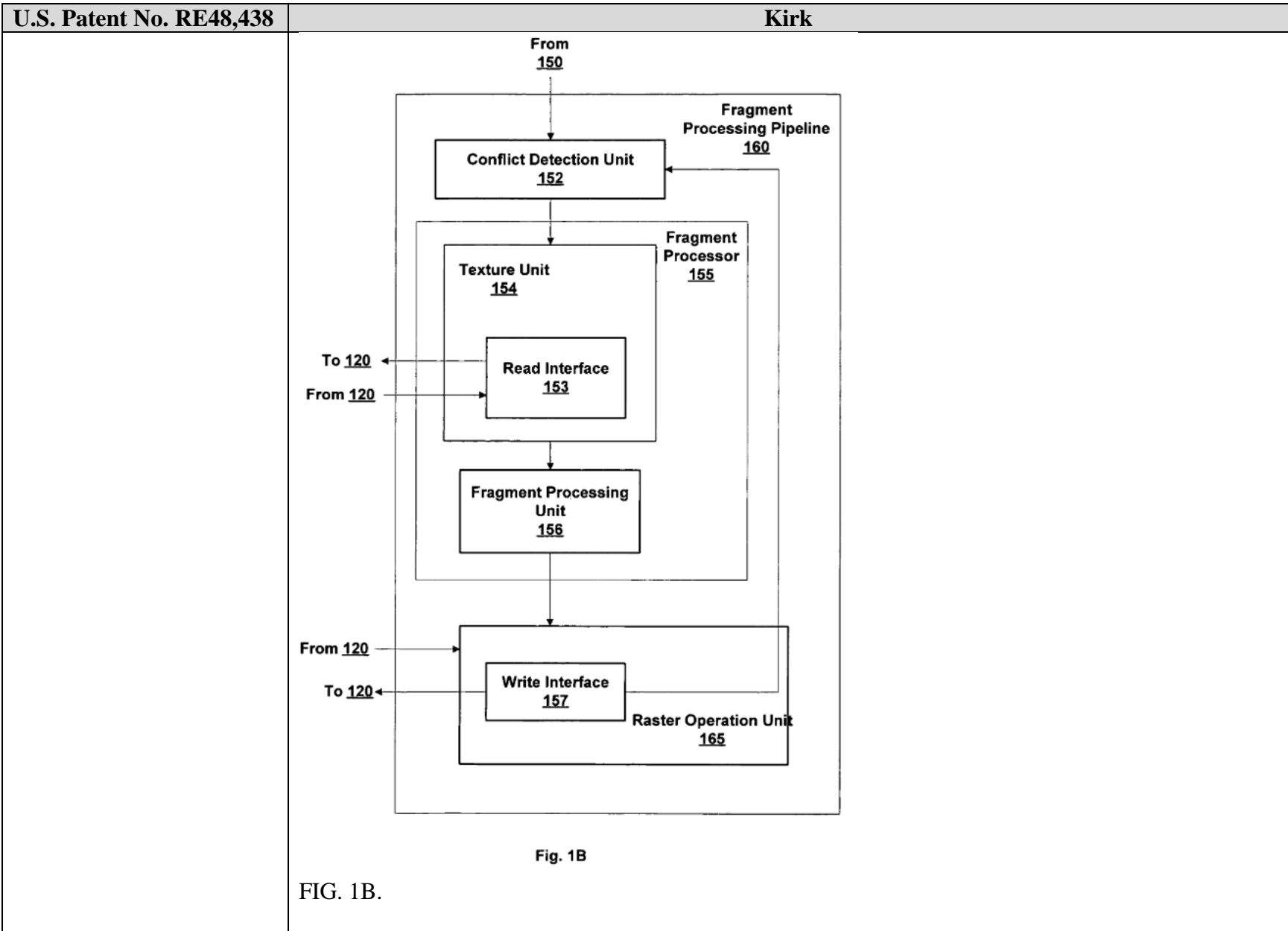
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 10:64–11:41.</p>

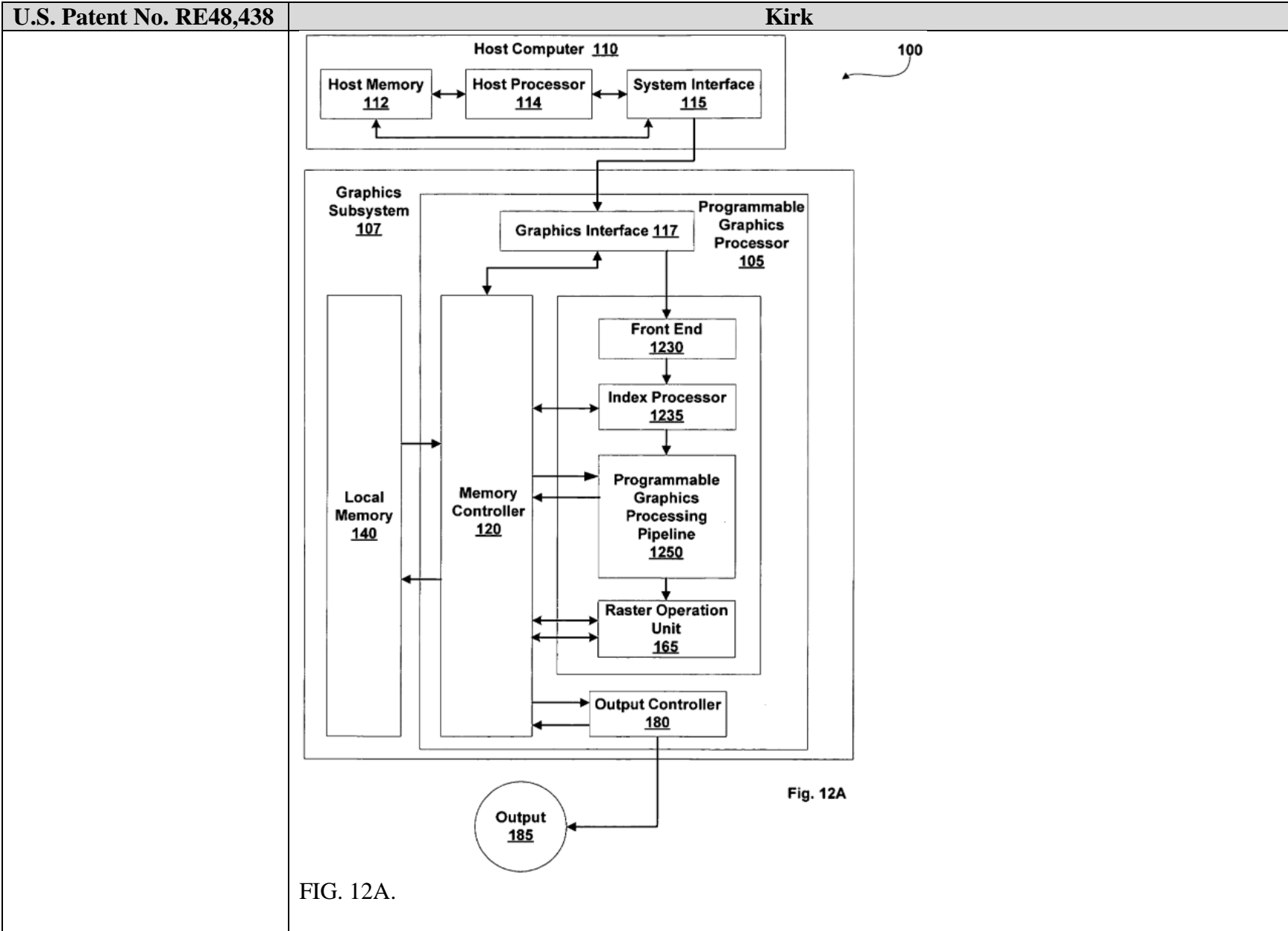
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

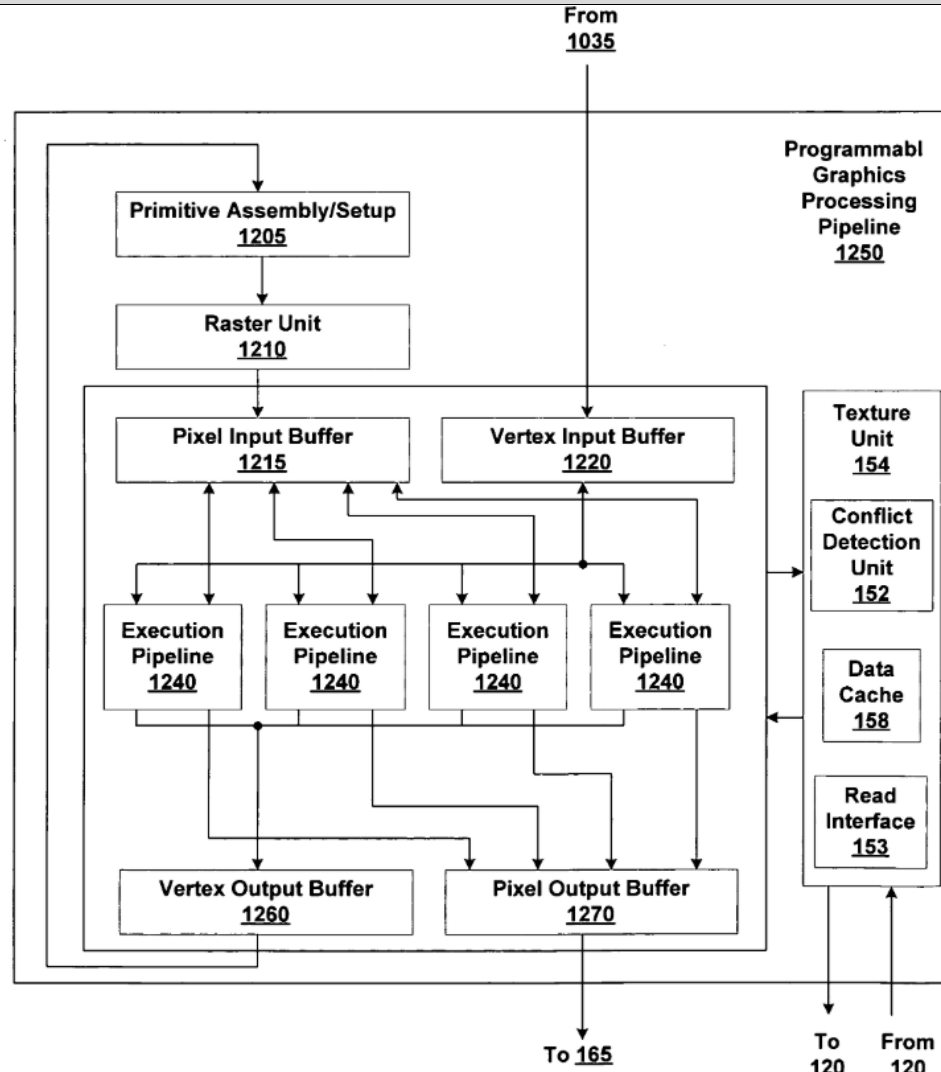


Fig. 12B

FIG. 12B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[c] “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory, the accelerator comprising:”</p>	<p>Kirk discloses “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory.” <i>See, e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses an accelerator (e.g., Graphics Subsystem 107), coupled to a central processing unit (e.g., Host Processor 114) and main memory (e.g., Host Memory 112) via a bus, that receives input data (e.g., which the CPU can access and send to Graphics Subsystem 107 for processing) from the main memory:</i></p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline of FIG. 1A in accordance with one or more aspects of the present invention. 2:14–21.</p> <p>FIG. 12A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem.</p> <p>FIGS. 12B and 12C are block diagrams of exemplary embodiments of the Programmable Graphics Processing Pipeline of FIG. 12A in accordance with one or more aspects of the present invention. 3:1–8.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a Computing System generally designated 100 and including a Host Computer 110 and a Graphics Subsystem 107. Computing</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>System 100 may be a desktop computer, server, laptop computer, palm-sized computer, tablet computer, game console, cellular telephone, computer-based simulator, or the like. Host computer 110 includes Host Processor 114 that may include a system memory controller to interface directly to Host Memory 112 or may communicate with Host Memory 112 through a System Interface 115. System Interface 115 may be an I/O (input/output) interface or a bridge device including the system memory controller to interface directly to Host Memory 112. Examples of System Interface 115 known in the art include Intel® Northbridge and Intel® Southbridge. 3:16–31.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>3:47–67.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like.</p> <p>4:21–42.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data.</p> <p>4:57–5:11.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p>

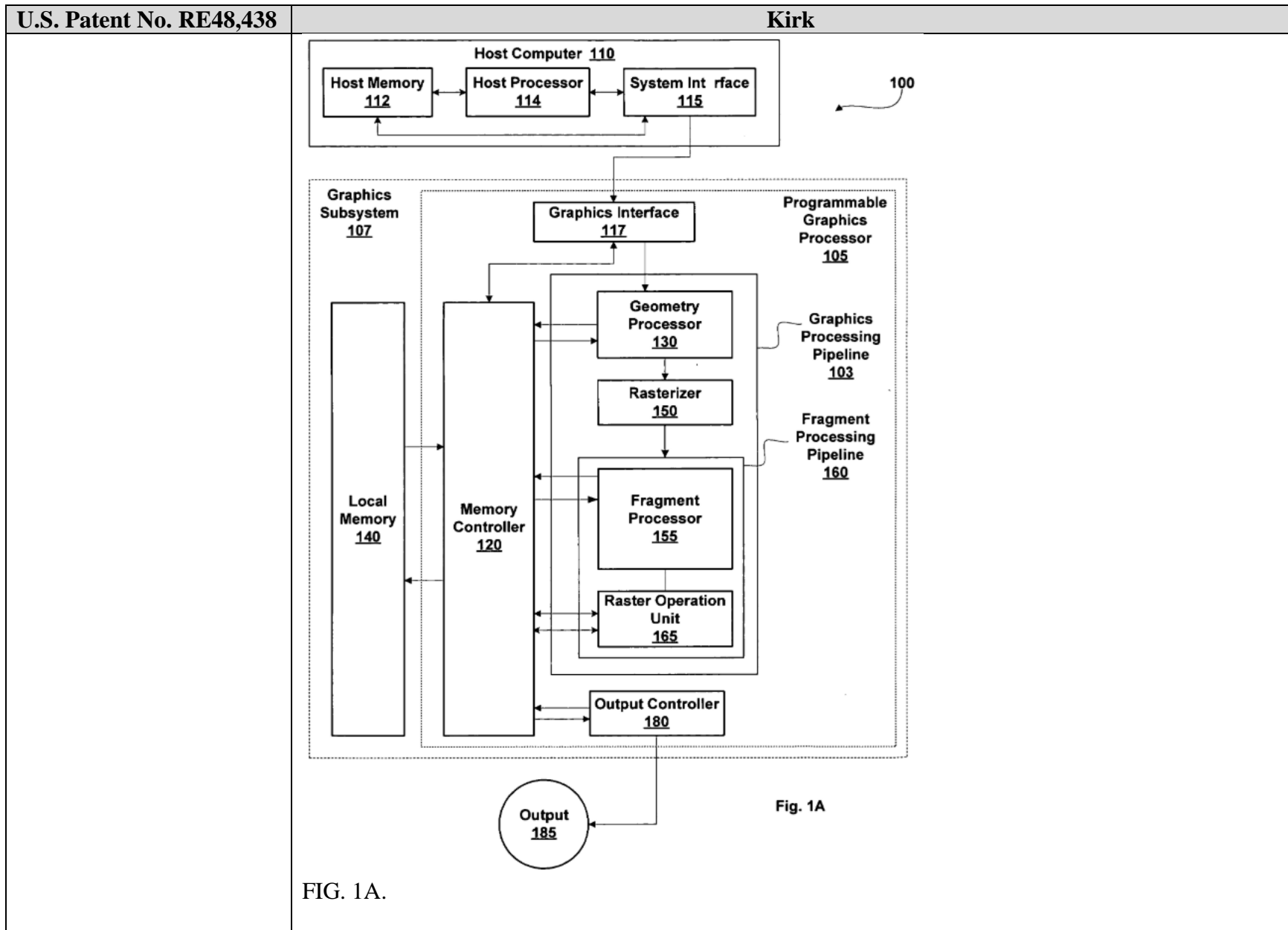
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>In step 209 Conflict Detection Unit 152 determines if a RAW position conflict exists for the position associated with the second fragment, and, if so, in step 211 Conflict Detection Unit 152 locks processing of the second fragment. Locking a fragment prevents any processing of the fragment requiring source data that is not yet available due to a RAW position conflict. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory. Step 214 may be completed several, even hundreds of clock cycles after step 205. Raster Operation Unit 165 outputs the write position information to Fragment Processor 155 confirming that the write is complete. In one embodiment the write is considered complete when the write request is output from Memory Controller 120 to Local Memory 140 or to Host Memory 112 via Graphics Interface 117. In another embodiment the write is considered complete when the write request is output from Raster Operation Unit 165 to Memory Controller 120. In step 217 Fragment Processing Pipeline 160 unlocks the second fragment and proceeds to step 220. In step 220 Fragment Processor 155 begins shading the second fragment as specified by the shader. 7:8–29.</p> <p>FIG. 5 is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160, including a Data Cache 158. Fragment data received by Fragment Processing Pipeline 160 from Rasterizer 150 is processed according to the fragment program instructions and Conflict Detection Unit 152 determines if a RAW conflict exists for each fragment. Conflict Detection Unit 152 outputs fragment data and the fragment program instructions to Fragment Processor 155. A Read Interface 153 within Texture Unit 154 reads additional fragment program instructions and buffer data from Data Cache 158. When the additional fragment program instructions or the buffer data are not available in Data Cache 158, i.e., there is a “cache miss”, the additional fragment program instructions or the buffer data is read from either Local Memory 140 or Host Memory 112, via Memory Controller 120 and optionally stored in Data Cache 158 and output by Read Interface 153. When a location is entered in Conflict Detection Unit 152 for a pending write, Conflict Detection Unit 152 determines if data stored in the location is available in an entry in Data Cache 158 and if so, invalidates the entry in Data Cache 158 as described further herein. In an alternate embodiment, entries in Data Cache 158 containing data read from a location that is entered in Conflict Detection Unit 152 are invalidated by Read Interface 153.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data</p>

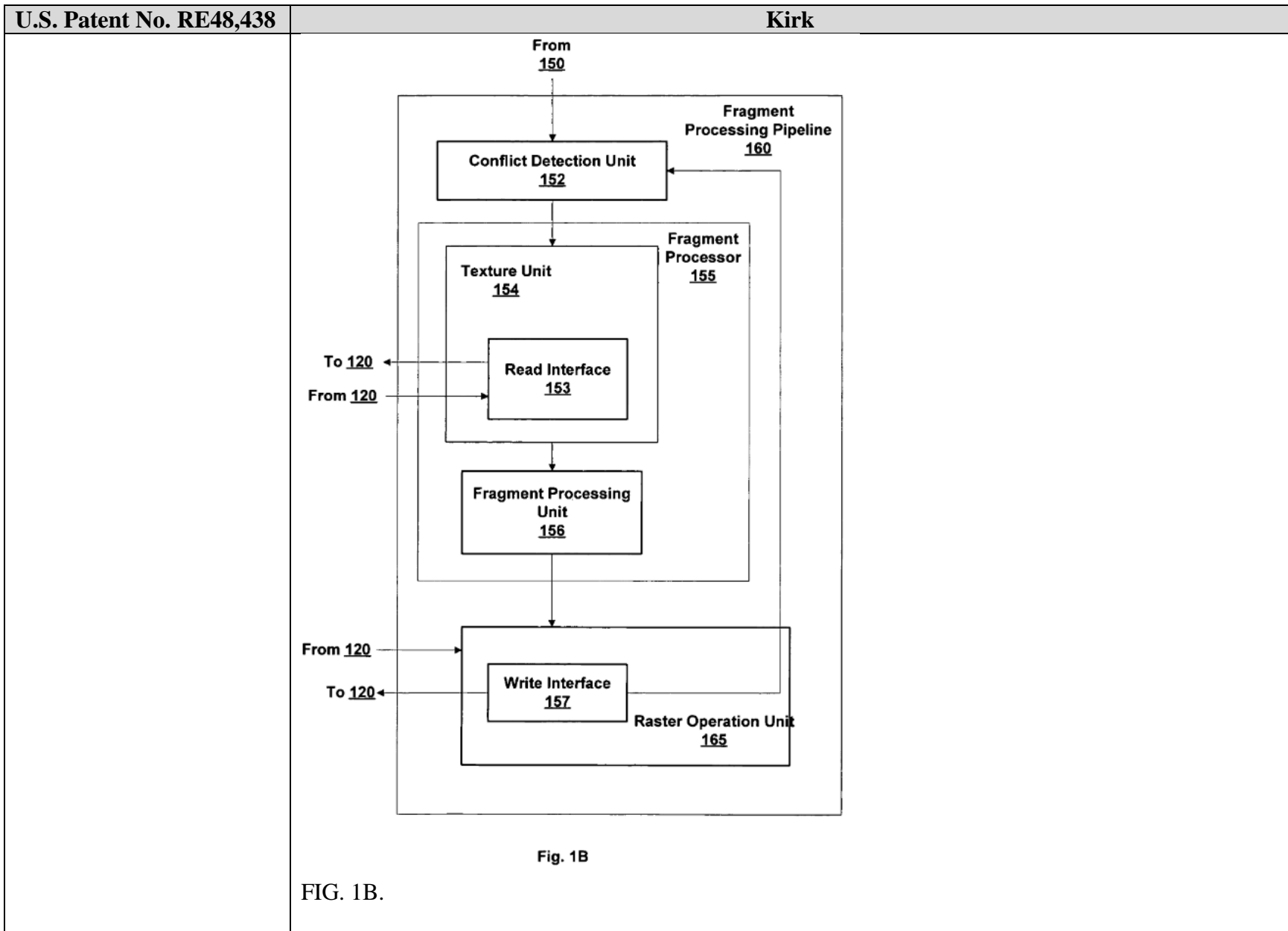
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 10:64–11:41.</p>

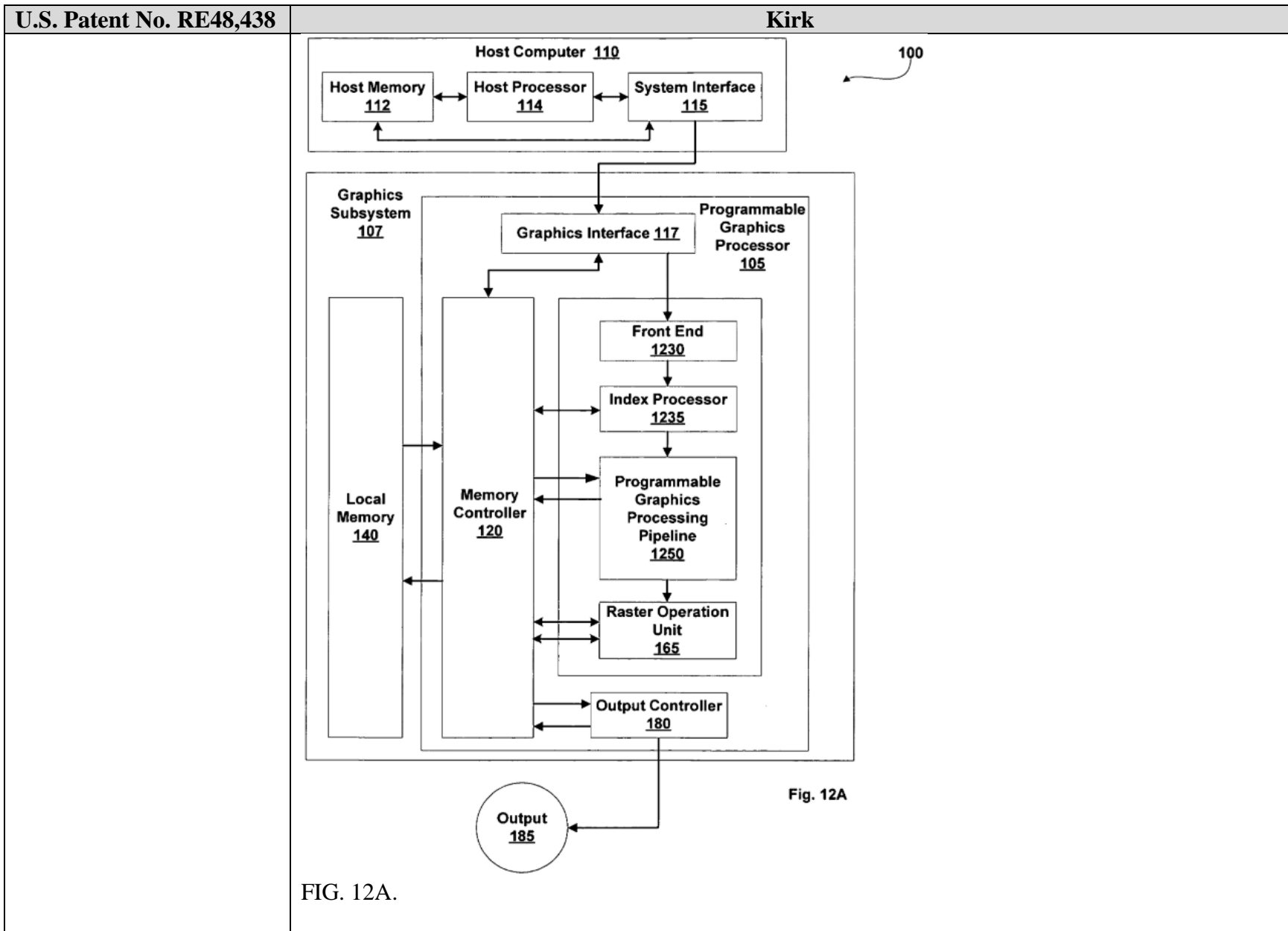
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

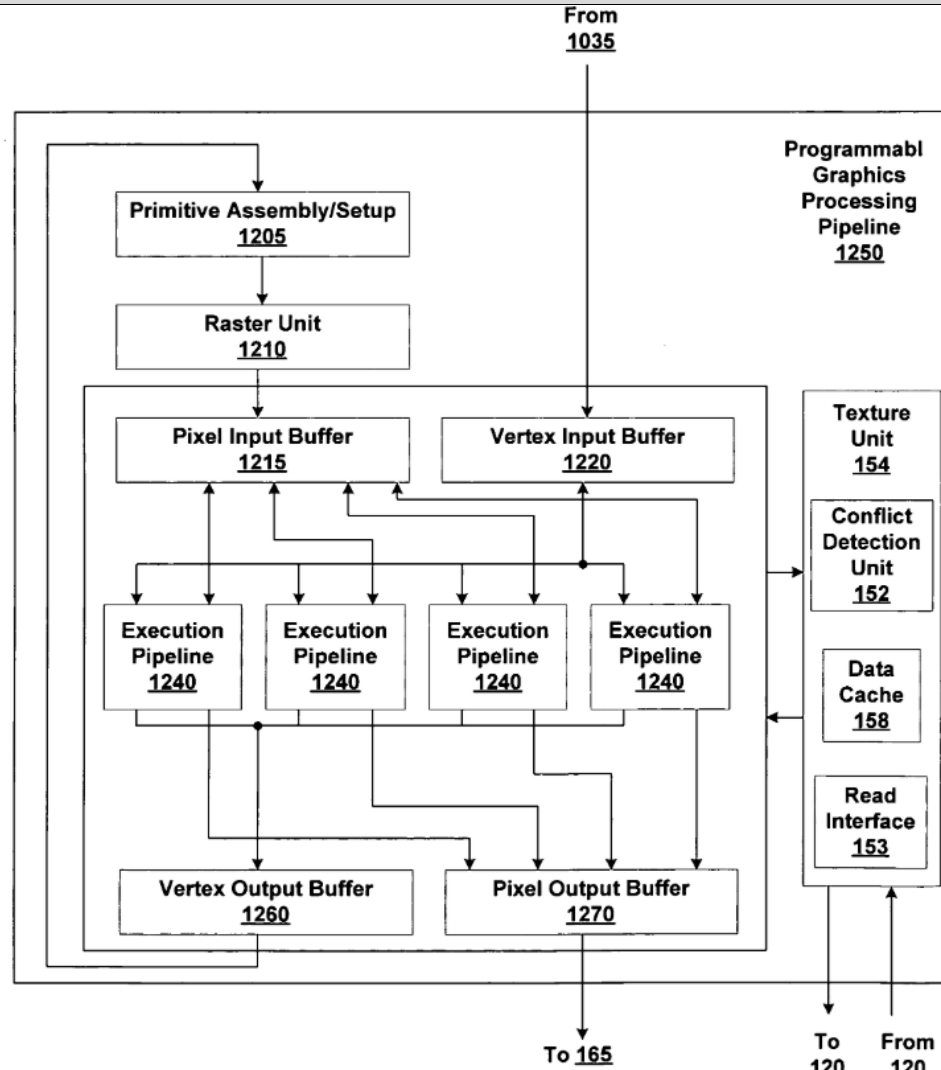


Fig. 12B

FIG. 12B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[c][i] “at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, the sequence of computations representing an artificial neural network, intermediate computations in the sequence of computations representing respective layers of the artificial neural network and yielding intermediate results; and”</p>	<p>Kirk discloses “at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, the sequence of computations representing an artificial neural network, intermediate computations in the sequence of computations representing respective layers of the artificial neural network and yielding intermediate results.” See <i>e.g.</i>:</p> <p><i>As a non-limiting example, Kirk discloses at least one graphics processing unit (e.g., Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250) that can perform computations (e.g., graphics programs and/or threads) on input data to generate output data (e.g., data produced by the processing pipeline, which may be written to a buffer in graphics memory to be read from during a subsequent pass):</i></p> <p>Apparatuses and methods for detecting position conflicts during fragment processing are described. Prior to executing a program on a fragment, a conflict detection unit, within a fragment processor checks if there is a position conflict indicating a RAW (read after write) hazard may exist. A RAW hazard exists when there is a pending write to a destination location that source data will be read from during execution of the program. When the fragment enters a processing pipeline, each destination location that may be written during the processing of the fragment is entered in conflict detection unit. During processing, the conflict detection unit is updated when a pending write to a destination location is completed.</p> <p>Abstract.</p> <p>For the foregoing reasons, it is desirable to write to a buffer and read from the buffer without flushing the shading pipeline between the write and read.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>1:32–34.</p> <p>Various embodiments of the invention include an application programming interface for a programmable graphics processor. The application programming interface includes one or more program instruction to configure a fragment processor within the programmable graphics processor to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved.</p> <p>Various embodiments of a method of the invention include processing fragment program instructions. A pixel load instruction including a source address corresponding to a location within the buffer is received. A write to the source address is determined to be pending. Data stored in the location corresponding to the source address is read after the write to the source address is complete.</p> <p>Various embodiments of a method of the invention include a fragment program for processing fragment data in a fragment processing pipeline. The fragment program includes a fragment program instruction to write a destination location in a buffer and a fragment program instruction to read the destination location in the buffer, without an intervening instruction to flush the fragment processing pipeline.</p> <p>Various embodiments of the invention include a computer program product having a computer readable medium having computer program instructions recorded thereon. The computer program product includes a fragment program for execution by a fragment processing pipeline. The fragment program includes a fragment program instruction to write a position in a buffer and a fragment program instruction to read the position in the buffer, without an intervening instruction to flush the fragment processing pipeline.</p> <p>1:40–2:4.</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline of FIG. 1A in accordance with one or more aspects of the present invention.</p> <p>2:14–21.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>FIG. 12A is a block diagram of an exemplary embodiment of a respective computer system in accordance with one or more aspects of the present invention including a host computer and a graphics subsystem.</p> <p>FIGS. 12B and 12C are block diagrams of exemplary embodiments of the Programmable Graphics Processing Pipeline of FIG. 12A in accordance with one or more aspects of the present invention. 3:1–8.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>Data 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.” 4:16–20</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>Data processed by Geometry Processor 130 and program instructions are passed from Geometry Processor 130 to a Rasterizer 150. Rasterizer 150 is a sampling unit that processes graphics primitives and generates sub-primitive data, such as pixel data or fragment data, including coverage data. Coverage data indicates which sub-pixel sample positions within a pixel are “covered” by a fragment formed by the intersection of the pixel and a primitive. Graphics primitives include geometry, such as points, lines, triangles, quadrilaterals, meshes, surfaces, and the like. Rasterizer 150 converts graphics primitives into sub-primitive data, performing scan conversion on the data processed by Geometry Processor 130. Rasterizer 150 outputs fragment data and fragment program instructions to Fragment Processing Pipeline 160. 4:43–57.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>FIGS. 2A, 2B, and 2C illustrate embodiments of methods of detecting and avoiding position conflicts in accordance with one or more aspects of the present invention. FIG. 2A illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. In step 205 Fragment Processing Pipeline 160 begins shading the first fragment as specified by a fragment program, producing a shaded first fragment, and outputs the shaded first fragment to Raster Operation Unit 165. Persons skilled in the art will recognize that depending on the complexity of the fragment program or the depth of a shading pipeline, several clocks cycles, even hundreds of clock cycles may pass before the shaded first fragment is produced. In step 207 Fragment Processing Pipeline 160 receives a second fragment associated with the position within the buffer. To produce a shaded second fragment, the fragment program specifies reading the shaded first fragment as source data. 6:56–7:7.</p> <p>If, in step 209 Conflict Detection Unit 152 determines a RAW position conflict does not exist for the position associated with the second fragment, then in step 220 Fragment Processor 155 begins shading the second fragment as specified by the fragment program producing a shaded second fragment. In step 222 Fragment Processor 155 receives one or more additional fragments, each fragment associated with a position for which a RAW position conflict does not exist. Fragment Processor 155 shades the one or more additional fragments. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory and outputs the write position information to Conflict Detection Unit 152 confirming that the write is complete. 7:30–43.</p> <p>FIG. 2B illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading including the steps illustrated in FIG. 2A. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. The fragment program specifies writing a shaded first fragment to the position within the buffer. In step 203 Conflict Detection Unit 152 receives the position. In one embodiment the position is represented as a pair of coordinates, e.g., (x,y), (s,t), (u,v), and the like, and the coordinates or portions of the coordinates are stored in Conflict Detection Unit 152. The coordinates may be represented relative to a buffer or relative to a display. Coordinates represented within a buffer may be converted into</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>coordinates within a display, e.g., screen coordinates, by applying coordinate offsets based on a position of the buffer within the display. In another embodiment the position is represented as an address for a location in graphics memory. In yet another embodiment the position includes a buffer identifier specifying which of several buffers the position is associated with. In still another embodiment, Conflict Detection Unit 152 identifies a region including the location and stores data, e.g. one or more bits, corresponding to the region. A region may represent several positions, where the positions may correspond to a region of an image, a region of an output buffer, a sequence of physical memory addresses in graphics memory, or the like. Conflict Detection Unit 152 may store data for several regions, depending on a predetermined resolution of the positions to be tracked.</p> <p>In step 205 Fragment Processor 155 begins shading the first fragment, as specified by the fragment program producing a shaded first fragment, several cycles or more later. The shaded first fragment is output to Raster Operation Unit 165. In step 207 Fragment Processing Pipeline 160 receives a second fragment associated with the position within the buffer. To produce a shaded second fragment, the fragment program specifies reading the shaded first fragment as source data.</p> <p>7:55–8:16.</p> <p>FIG. 4B illustrates a method of processing graphics data including some of the steps shown in FIG. 4A. A fragment program specifies writing data to a location in a buffer to process a first fragment and reading the data from the location in the buffer to produce shaded fragment data for a second fragment without an intervening flush of Fragment Processor 155 or Fragment Processing Pipeline 160.</p> <p>In step 401 fragments are received by Fragment Processing Pipeline 160. In step 402 the location in the buffer to be written by the first fragment is entered in Conflict Detection Unit 152 (CDU). The second fragment is also associated with the location in the buffer, specifically the fragment program specifies using data read from the location (source data) to produce a shaded second fragment. Conflict Detection Unit 152 determines that a write to the location in the buffer is pending and does not initiate reading the location in the buffer. Steps 405, 409, 411, and 413 are completed as previously described in relation to FIG. 4A.</p> <p>10:39–56.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via</p>

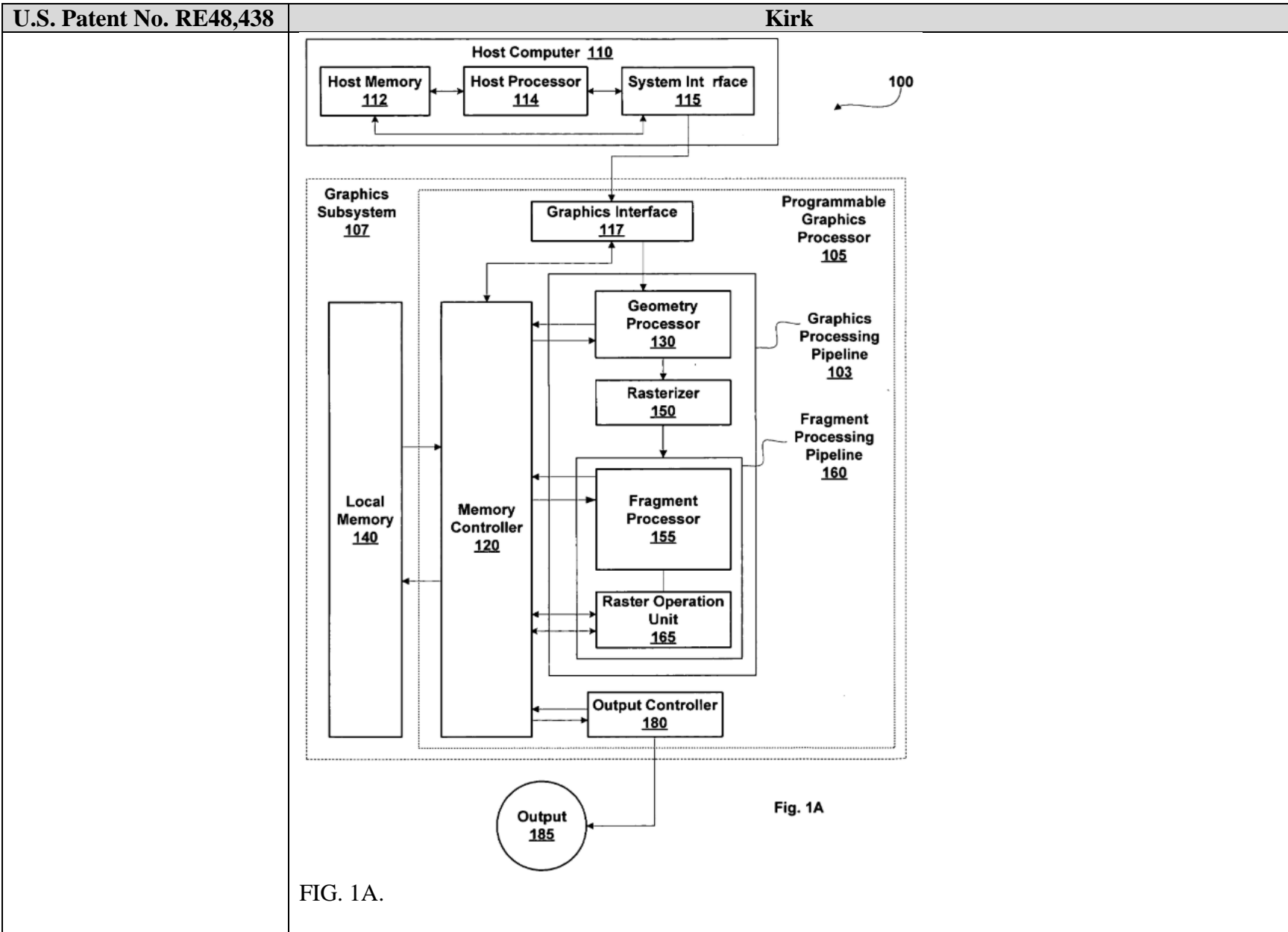
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 26:56–61.</p> <p>FIG. 12B is a block diagram of an exemplary embodiment of Programmable Graphics Processing Pipeline 1250 in accordance with one or more aspects of the present invention. Samples, such as surfaces, primitives, or the like, are received from Index Processor 1235 by Programmable Graphics Processing Pipeline 1250 and stored in a Vertex Input Buffer 1220 in a register file, FIFO (first in first out) memory, cache, or the like (not shown). The samples are broadcast to Execution Pipelines 1240,</p>

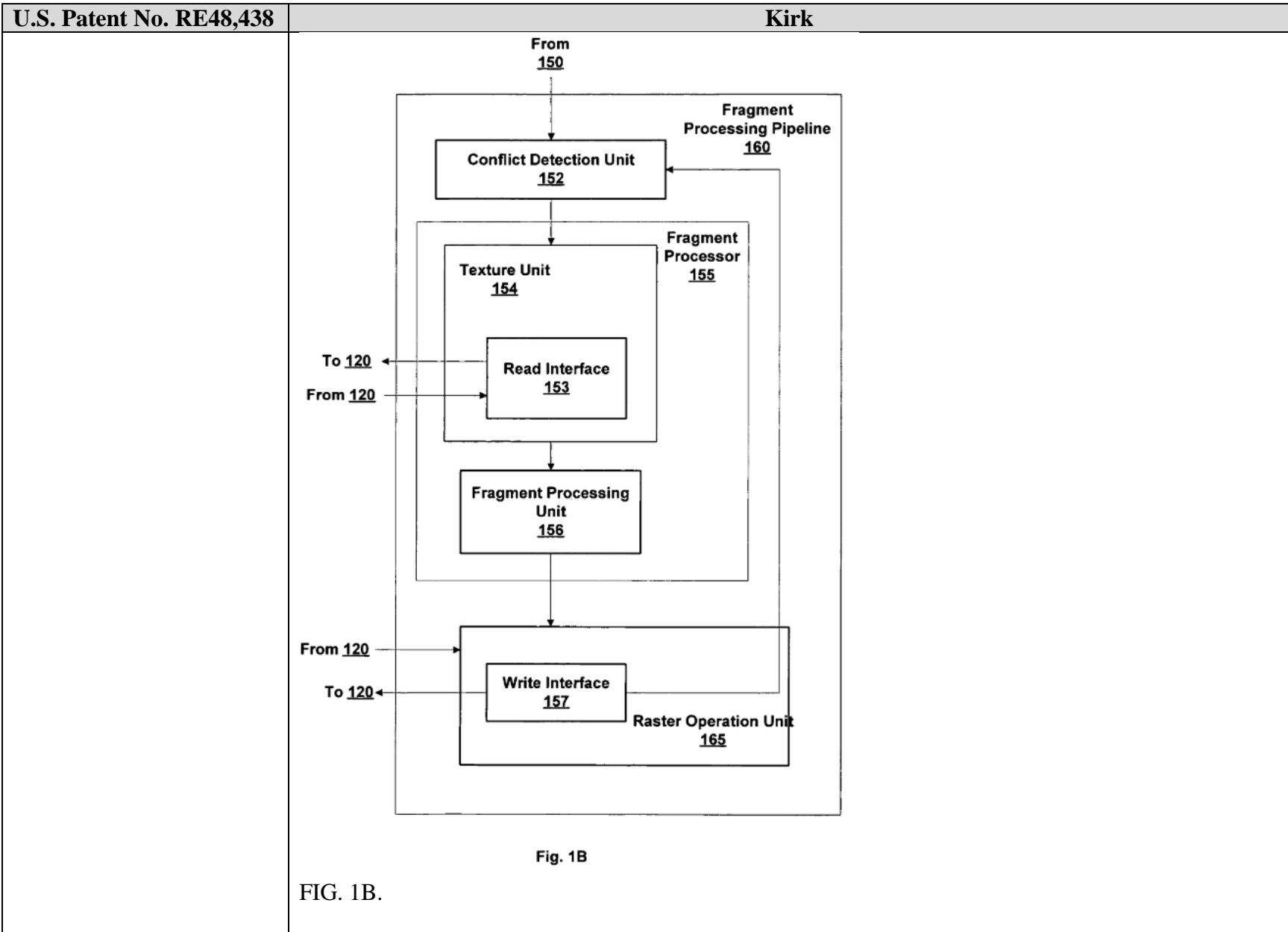
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>four of which are shown in FIG. 12B. An alternate embodiment may include either more or fewer Execution Pipelines 1240. Each Execution Pipeline 1240 includes at least one multithreaded processing unit. The samples output by Vertex Input Buffer 1220 may be processed by any one of the Execution Pipelines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available. 26:62–27:11.</p> <p>Execution Pipelines 1240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 1240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics data stored in buffers in graphics memory via Memory Controller 120. An optional Data Cache 158 within Texture Unit 154 is used to improve memory read performance by reducing read latency. In another alternate embodiment, a Texture Unit 154 is included in each Execution Pipeline 1240. In yet another alternate embodiment, program instructions are stored within Programmable Graphics Processing Pipeline 1250. 27:12–28.</p> <p>A graphics program (vertex program or fragment program) is executed within one or more Execution Pipelines 1240 as a plurality of threads where each vertex or fragment to be processed by the program is assigned to a thread. Although threads share processing resources within Programmable Graphics Processing Pipeline 1250 and graphics memory, the execution of each thread proceeds in the one or more Execution Pipelines 1240 independent of any other threads. A RAW position conflict may exist when a fragment program specifies to write to a position in a buffer that the fragment program later specifies to read from. Likewise, a RAW position conflict may exist when a fragment program specifies to write to a position in a buffer that a subsequent fragment program specifies to read from. Furthermore, because threads are executed independently, RAW conflicts may exist when a thread executes a write to a position in a buffer that the thread or another thread executes a read from. 27:40–56.</p>

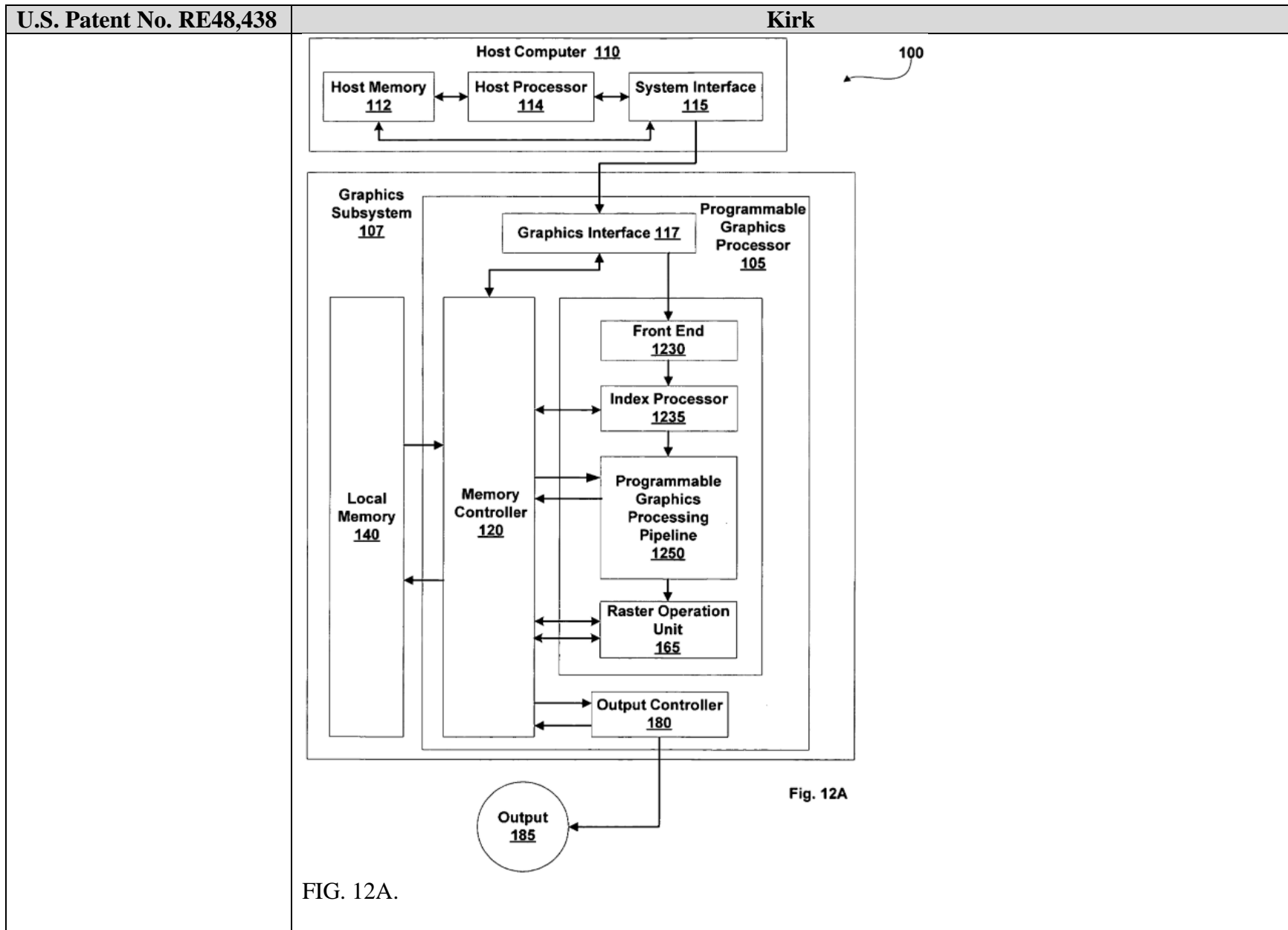
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

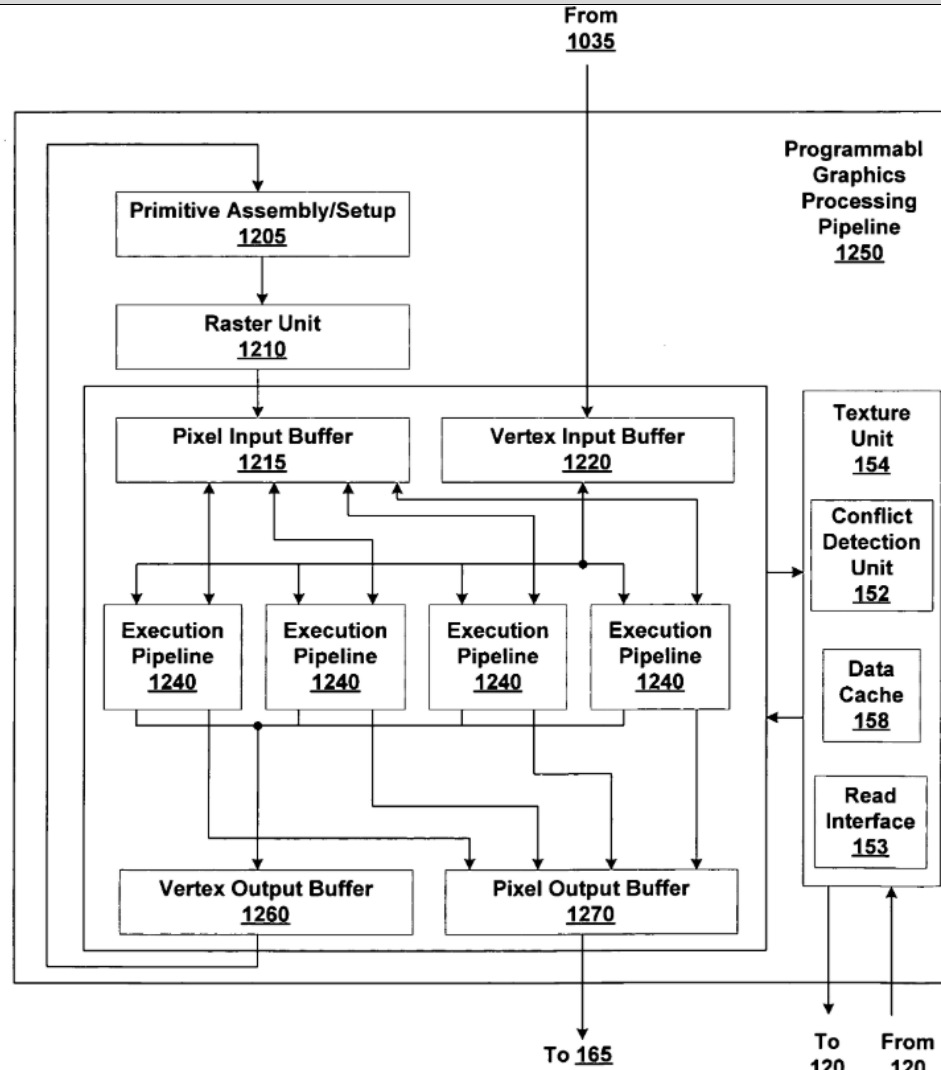


Fig. 12B

FIG. 12B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[c][ii] “accelerator memory, operably coupled to the at least one graphics processing unit, to store the results of the sequence of computations; and”</p>	<p>Kirk discloses “accelerator memory, operably coupled to the at least one graphics processing unit, to store the results of the sequence of computations.” <i>See e.g.:</i></p> <p><i>As a nonlimiting example, Kirk discloses accelerator memory (e.g., Local Memory 140) operably coupled to a graphics processing unit (e.g., Programmable Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250), which can store input and output data, such as the results of the sequences of computations performed in the graphics processing unit:</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>In various embodiments Memory Controller 120, Local Memory 140, and Geometry Processor 130 are configured such that data generated at various points along Graphics Processing Pipeline 103 may be output via Raster Operation Unit 165 and provided to Geometry Processor 130 or Fragment Processor 155 as input. The output data is represented in one or more formats as specified by the codewords. For example, color data may be written as 16, 32, 64, or 128 bits per pixel fixed or floating-point RGBA (red, green, blue, and alpha) to be scanned out for display. As a specific example, four 16-bit floating-point components (RGBA) are combined forming 64 bits of color data for each fragment. The output data, e.g., color, depth, and other parameters, may be processed</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>according to a fragment program and stored in a buffer in graphics memory to be used as a texture map, e.g., shadow map, height field, stencil, and the like, by the fragment program. Alternatively, color and depth output data may be written to a buffer, and later read and processed by Raster Operation Unit 165 to generate the final pixel data prior to being scanned out for display via Output Controller 180.</p> <p>For example, Fragment Processing Pipeline 160 is configured by fragment program instructions to produce processed data and store the processed data in a buffer in Local Memory 140. The Fragment Processing Pipeline 160 is configured by the fragment program instructions to read and further process the processed data. For example, Fragment Processing Pipeline 160 may be configured to implement a modified depth buffer algorithm, e.g., sorting and maintaining more than one depth value for each pixel. A modified depth buffer algorithm may be used to implement correct transparency by rendering fragments in back to front order while applying transparency blending. 5:12–45.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–60:3.</p> <p>Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:34–55.</p> <p>In step 209 Conflict Detection Unit 152 determines if a RAW position conflict exists for the position associated with the second fragment, and, if so, in step 211 Conflict Detection Unit 152 locks processing of the second fragment. Locking a fragment prevents any processing of the fragment requiring source data that is not yet available due to a RAW position conflict. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory. Step 214 may be completed several, even hundreds of clock cycles after step 205. Raster Operation Unit 165 outputs the write position information to Fragment Processor 155 confirming that the write is complete. In one embodiment the write is considered complete when the write request is output from Memory Controller 120 to Local Memory 140 or to Host Memory 112 via Graphics Interface 117. In another embodiment the write is considered complete when the write request is output from Raster Operation Unit 165 to Memory Controller 120. In step 217 Fragment Processing Pipeline 160 unlocks the second fragment and proceeds to step 220. In step 220 Fragment Processor 155 begins shading the second fragment as specified by the shader. 7:8–29.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p>

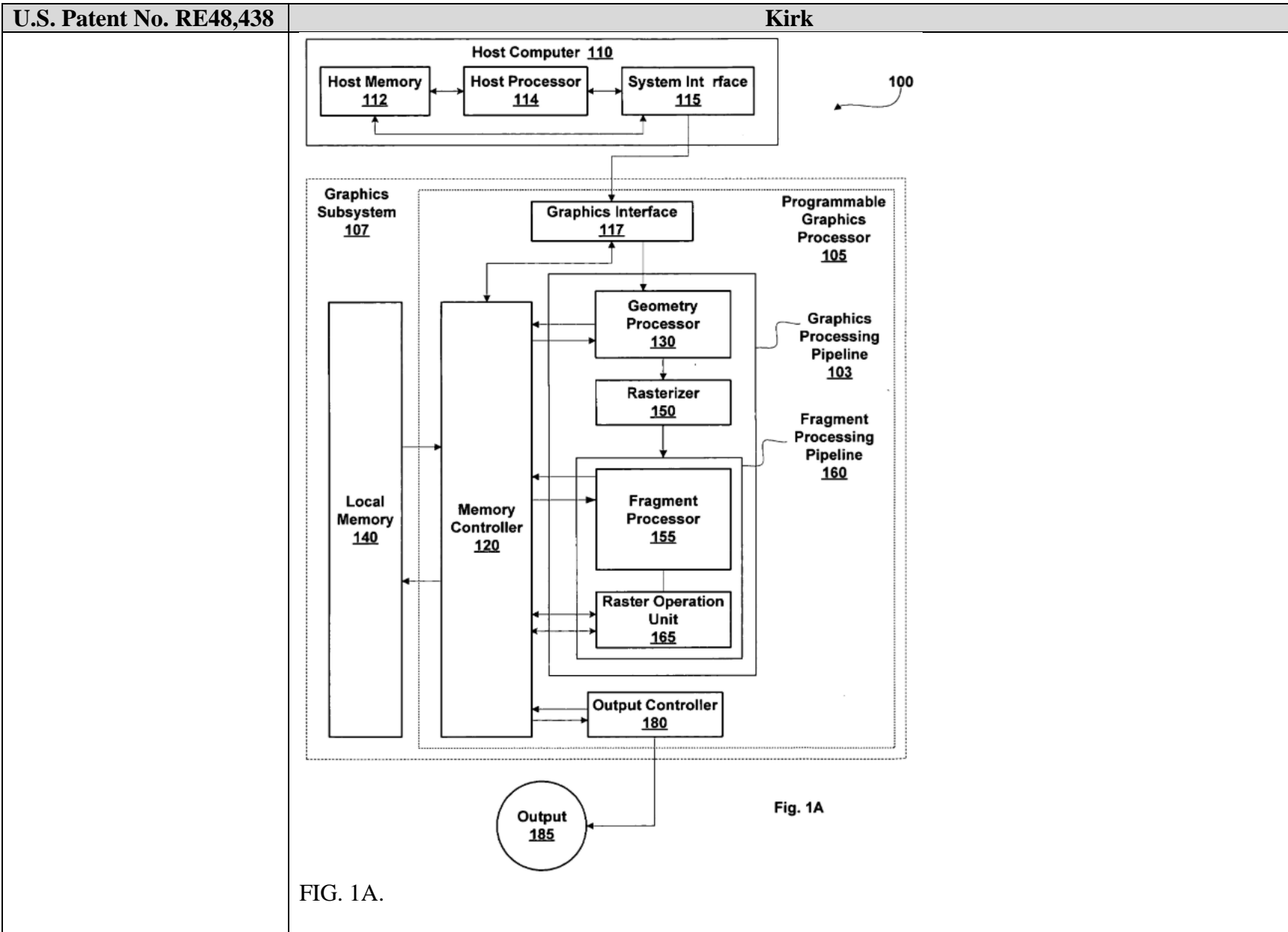
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 26:56–61.</p>

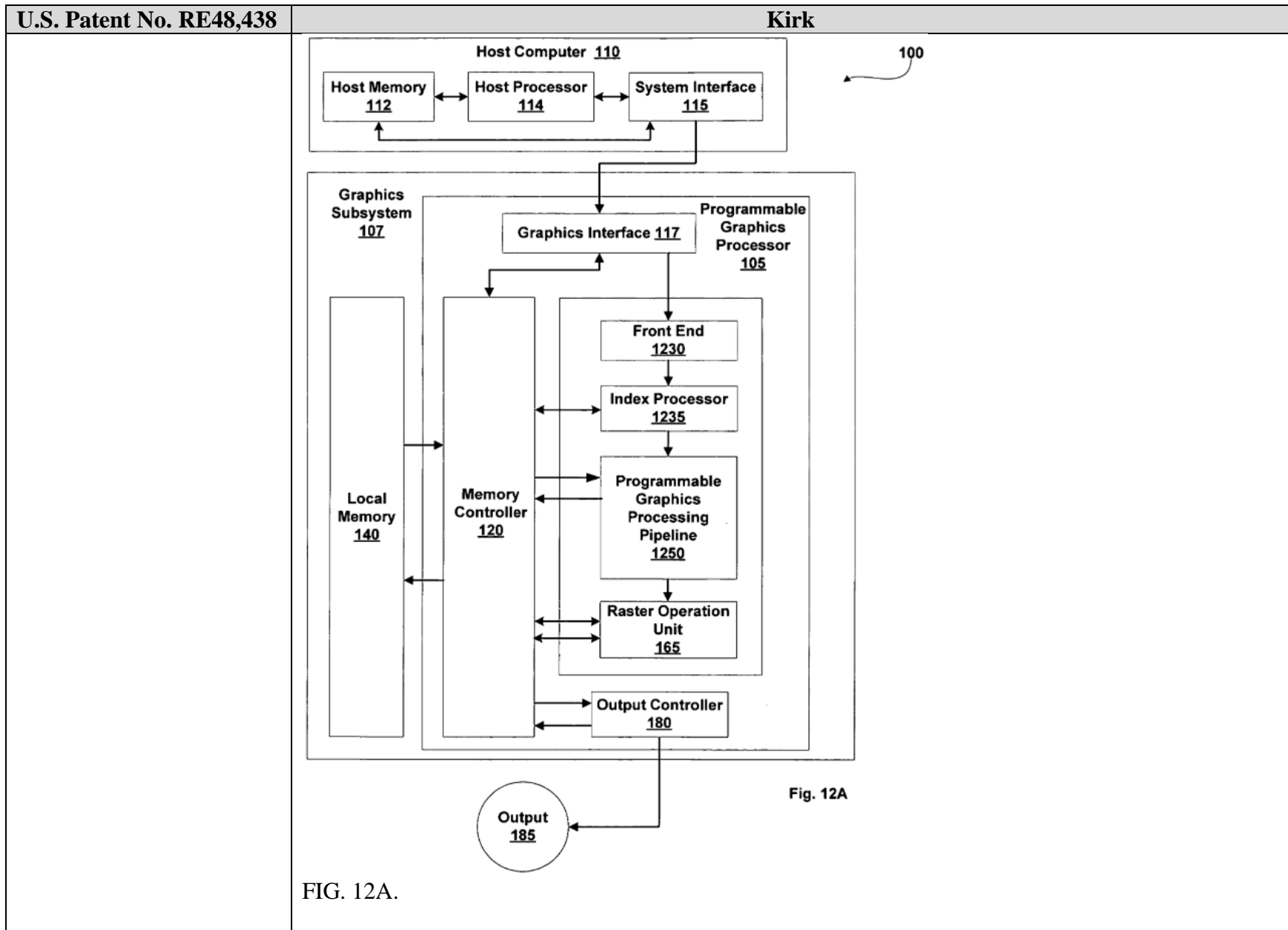
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Execution Pipelines 1240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 1240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics data stored in buffers in graphics memory via Memory Controller 120. An optional Data Cache 158 within Texture Unit 154 is used to improve memory read performance by reducing read latency. In another alternate embodiment, a Texture Unit 154 is included in each Execution Pipeline 1240. In yet another alternate embodiment, program instructions are stored within Programmable Graphics Processing Pipeline 1250. 27:12–28.</p> <p>Execution Pipelines 1240 output processed samples, such as vertices, that are stored in a Vertex Output Buffer 1260 in a register file, FIFO memory, cache, or the like (not shown). Processed vertices output by Vertex Output Buffer 1260 are received by a Primitive Assembly/Setup 1205. Primitive Assembly/Setup 1205 calculates parameters, such as deltas and slopes, for rasterizing the processed vertices. Primitive Assembly/Setup 1205 outputs parameters and samples, such as vertices, to Raster Unit 1210. The Raster Unit 1210 performs scan conversion on samples and outputs fragments to a Pixel Input Buffer 1215. 27:29–39.</p> <p>Execution Pipelines 1240 are optionally configured using program instructions read by Texture Unit 154 such that data processing operations are performed in multiple passes through at least one multithreaded processing unit within Execution Pipelines 1240. 28:33–37.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d] “a controller, operably coupled to the at least one graphics processing unit and the accelerator memory,”</p>	<p>Kirk discloses “a controller, operably coupled to the at least one graphics processing unit and the accelerator memory.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses a controller (e.g., Graphics Interface 117, Front End 1230, and/or Index Processor 1235), which is coupled to at least one graphics processing unit (e.g., Programmable Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250) and accelerator memory (e.g., Local Memory 140):</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>In various embodiments Memory Controller 120, Local Memory 140, and Geometry Processor 130 are configured such that data generated at various points along Graphics Processing Pipeline 103 may be output via Raster Operation Unit 165 and provided to Geometry Processor 130 or Fragment Processor 155 as input. The output data is represented in one or more formats as specified by the codewords. For example, color data may be written as 16, 32, 64, or 128 bits per pixel fixed or floating-point RGBA (red, green, blue, and alpha) to be scanned out for display. As a specific example, four 16-bit floating-point components (RGBA) are combined forming 64 bits of color data for each fragment. The output data, e.g., color, depth, and other parameters, may be processed</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>according to a fragment program and stored in a buffer in graphics memory to be used as a texture map, e.g., shadow map, height field, stencil, and the like, by the fragment program. Alternatively, color and depth output data may be written to a buffer, and later read and processed by Raster Operation Unit 165 to generate the final pixel data prior to being scanned out for display via Output Controller 180.</p> <p>For example, Fragment Processing Pipeline 160 is configured by fragment program instructions to produce processed data and store the processed data in a buffer in Local Memory 140. The Fragment Processing Pipeline 160 is configured by the fragment program instructions to read and further process the processed data. For example, Fragment Processing Pipeline 160 may be configured to implement a modified depth buffer algorithm, e.g., sorting and maintaining more than one depth value for each pixel. A modified depth buffer algorithm may be used to implement correct transparency by rendering fragments in back to front order while applying transparency blending. 5:12–45.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>In step 209 Conflict Detection Unit 152 determines if a RAW position conflict exists for the position associated with the second fragment, and, if so, in step 211 Conflict Detection Unit 152 locks processing of the second fragment. Locking a fragment prevents any processing of the fragment requiring source data that is not yet available due to a RAW position conflict. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory. Step 214 may be completed several, even hundreds of clock cycles after step 205. Raster Operation Unit 165 outputs the write position information to Fragment Processor 155 confirming that the write is complete. In one embodiment the write is considered complete when the write request is output from Memory Controller 120 to Local Memory 140 or to Host Memory 112 via Graphics Interface 117. In another embodiment the write is considered complete when the write request is</p>

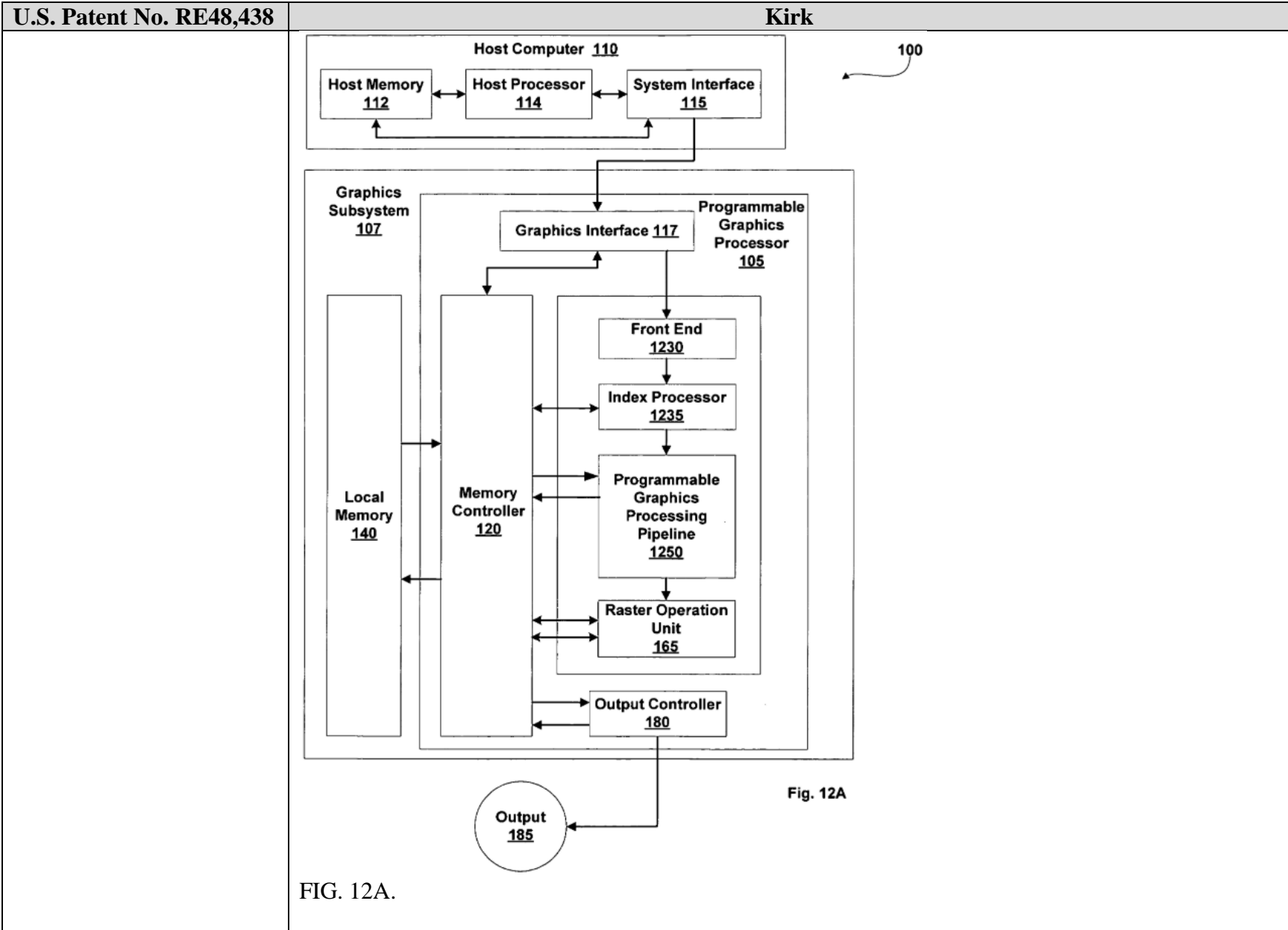
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>output from Raster Operation Unit 165 to Memory Controller 120. In step 217 Fragment Processing Pipeline 160 unlocks the second fragment and proceeds to step 220. In step 220 Fragment Processor 155 begins shading the second fragment as specified by the shader. 7:8–29.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>FIG. 12B is a block diagram of an exemplary embodiment of Programmable Graphics Processing Pipeline 1250 in accordance with one or more aspects of the present invention. Samples, such as surfaces, primitives, or the like, are received from Index Processor 1235 by Programmable Graphics</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Processing Pipeline 1250 and stored in a Vertex Input Buffer 1220 in a register file, FIFO (first in first out) memory, cache, or the like (not shown). The samples are broadcast to Execution Pipelines 1240, four of which are shown in FIG. 12B. An alternate embodiment may include either more or fewer Execution Pipelines 1240. Each Execution Pipeline 1240 includes at least one multithreaded processing unit. The samples output by Vertex Input Buffer 1220 may be processed by any one of the Execution Pipelines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available.</p> <p>26:62–27:11.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][i] “to initialize textures and shaders in the accelerator memory for performing the sequence of computations,”</p>	<p>Kirk discloses “to initialize textures and shaders in the accelerator memory for performing the sequence of computations.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses initializing textures (e.g., data) and shaders (e.g., program instructions) in accelerator memory (e.g., Local Memory 140) for performing computations:</i></p> <p>Various embodiments of the invention include an application programming interface for a programmable graphics processor. The application programming interface includes one or more program instruction to configure a fragment processor within the programmable graphics processor to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved.</p> <p>Various embodiments of a method of the invention include processing fragment program instructions. A pixel load instruction including a source address corresponding to a location within the buffer is received. A write to the source address is determined to be pending. Data stored in the location corresponding to the source address is read after the write to the source address is complete.</p> <p>Various embodiments of a method of the invention include a fragment program for processing fragment data in a fragment processing pipeline. The fragment program includes a fragment program instruction to write a destination location in a buffer and a fragment program instruction to read the destination location in the buffer, without an intervening instruction to flush the fragment processing pipeline.</p> <p>Various embodiments of the invention include a computer program product having a computer readable medium having computer program instructions recorded thereon. The computer program product includes a fragment program for execution by a fragment processing pipeline. The fragment program includes a fragment program instruction to write a position in a buffer and a fragment</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>program instruction to read the position in the buffer, without an intervening instruction to flush the fragment processing pipeline. 1:40–2:4.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group.</p> <p>It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible. 3:12–67.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>Data processed by Geometry Processor 130 and program instructions are passed from Geometry Processor 130 to a Rasterizer 150. Rasterizer 150 is a sampling unit that processes graphics primitives and generates sub-primitive data, such as pixel data or fragment data, including coverage data. Coverage data indicates which sub-pixel sample positions within a pixel are “covered” by a fragment formed by the intersection of the pixel and a primitive. Graphics primitives include geometry, such as points, lines, triangles, quadrilaterals, meshes, surfaces, and the like. Rasterizer 150 converts graphics primitives into sub-primitive data, performing scan conversion on the data processed by Geometry</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Processor 130. Rasterizer 150 outputs fragment data and fragment program instructions to Fragment Processing Pipeline 160. 4:43–57.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:58–5:11.</p> <p>In various embodiments Memory Controller 120, Local Memory 140, and Geometry Processor 130 are configured such that data generated at various points along Graphics Processing Pipeline 103 may be output via Raster Operation Unit 165 and provided to Geometry Processor 130 or Fragment Processor 155 as input. The output data is represented in one or more formats as specified by the codewords. For example, color data may be written as 16, 32, 64, or 128 bits per pixel fixed or floating-point RGBA (red, green, blue, and alpha) to be scanned out for display. As a specific example, four 16-bit floating-point components (RGBA) are combined forming 64 bits of color data for each fragment. The output data, e.g., color, depth, and other parameters, may be processed according to a fragment program and stored in a buffer in graphics memory to be used as a texture map, e.g., shadow map, height field, stencil, and the like, by the fragment program. Alternatively, color and depth output data may be written to a buffer, and later read and processed by Raster Operation Unit 165 to generate the final pixel data prior to being scanned out for display via Output Controller 180.</p>

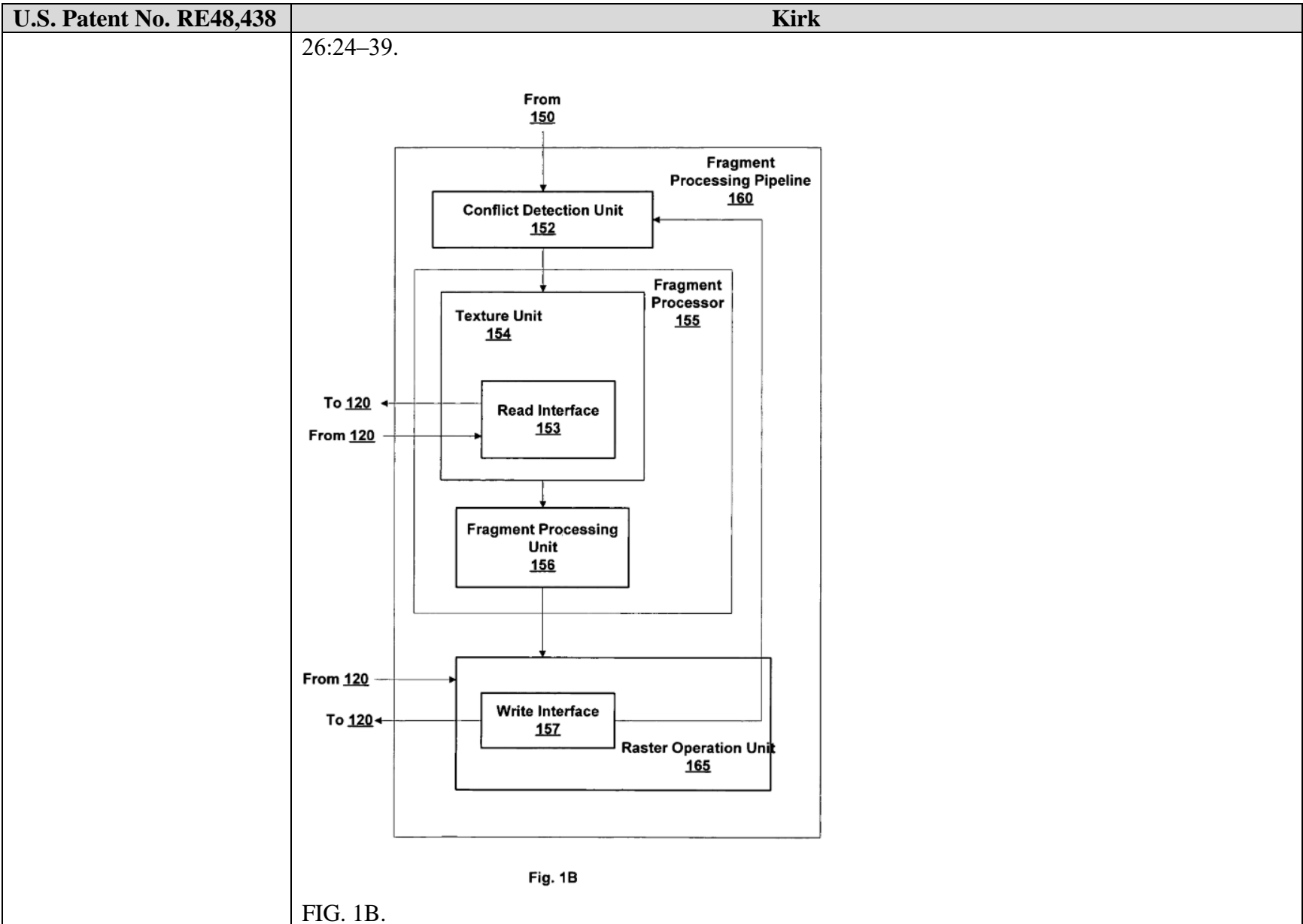
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>For example, Fragment Processing Pipeline 160 is configured by fragment program instructions to produce processed data and store the processed data in a buffer in Local Memory 140. The Fragment Processing Pipeline 160 is configured by the fragment program instructions to read and further process the processed data. For example, Fragment Processing Pipeline 160 may be configured to implement a modified depth buffer algorithm, e.g., sorting and maintaining more than one depth value for each pixel. A modified depth buffer algorithm may be used to implement correct transparency by rendering fragments in back to front order while applying transparency blending. 5:12–45.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments</p>

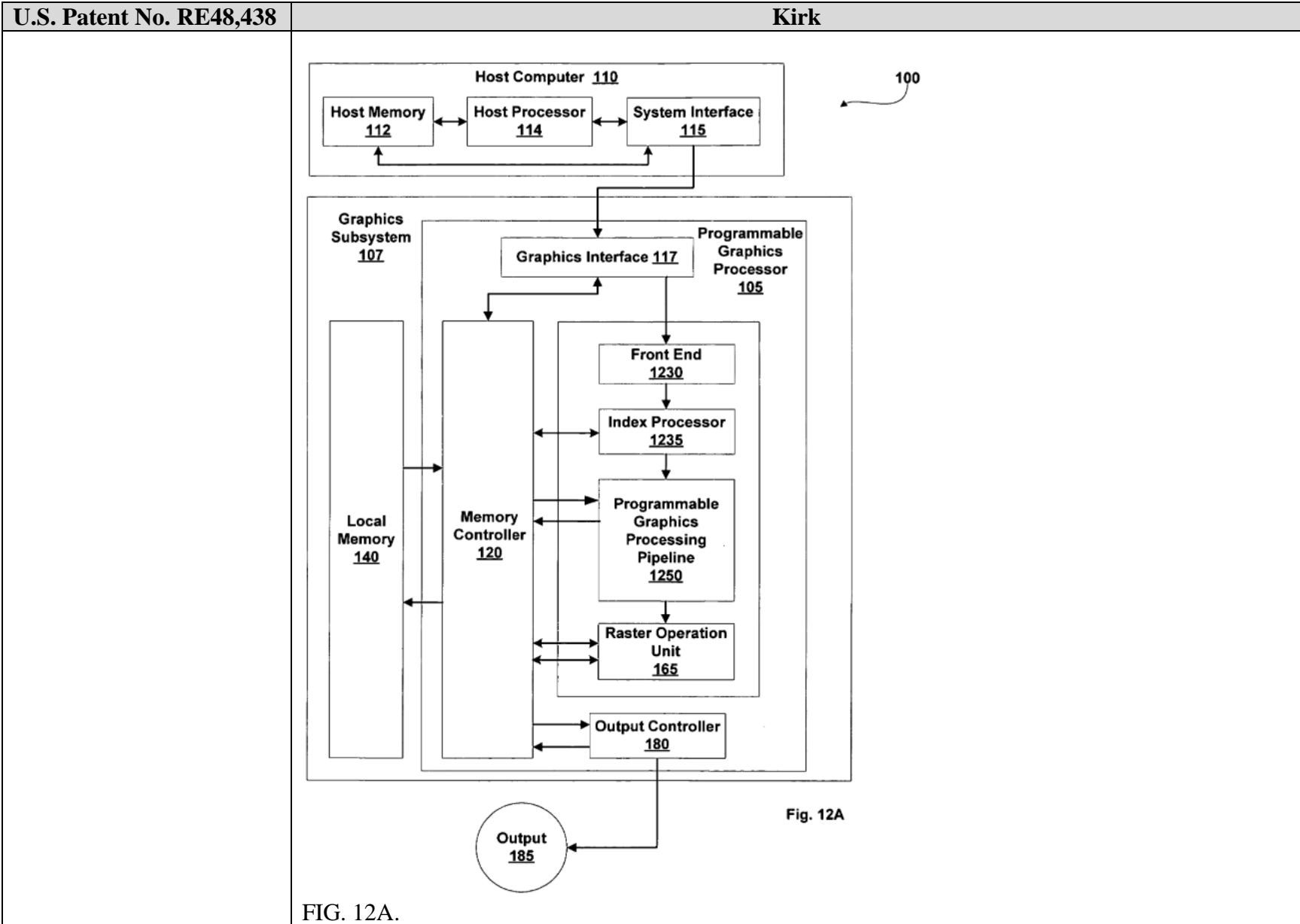
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][ii] “to control performance of the sequence of computations by the at least one graphics processing unit,”</p>	<p>Kirk discloses “to control performance of the sequence of computations by the at least one graphics processing unit.” <i>See e.g.</i>:</p> <p><i>As a non-limiting example, Kirk discloses that the controller (e.g., Graphics Interface 117, Front End 1230, and/or Index Processor 1235) controls performance (e.g., by issuing instructions) of the computations by the at least one graphics processing unit (e.g., Programmable Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250):</i></p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group.</p> <p>It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302 producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible.</p> <p>3:12–67.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>Data processed by Geometry Processor 130 and program instructions are passed from Geometry Processor 130 to a Rasterizer 150. Rasterizer 150 is a sampling unit that processes graphics primitives and generates sub-primitive data, such as pixel data or fragment data, including coverage data. Coverage data indicates which sub-pixel sample positions within a pixel are “covered” by a fragment formed by the intersection of the pixel and a primitive. Graphics primitives include geometry, such as points, lines, triangles, quadrilaterals, meshes, surfaces, and the like. Rasterizer 150 converts graphics primitives into sub-primitive data, performing scan conversion on the data processed by Geometry Processor 130. Rasterizer 150 outputs fragment data and fragment program instructions to Fragment Processing Pipeline 160. 4:43–57.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>FIG. 12B is a block diagram of an exemplary embodiment of Programmable Graphics Processing Pipeline 1250 in accordance with one or more aspects of the present invention. Samples, such as surfaces, primitives, or the like, are received from Index Processor 1235 by Programmable Graphics Processing Pipeline 1250 and stored in a Vertex Input Buffer 1220 in a register file, FIFO (first in first out) memory, cache, or the like (not shown). The samples are broadcast to Execution Pipelines 1240, four of which are shown in FIG. 12B. An alternate embodiment may include either more or fewer Execution Pipelines 1240. Each Execution Pipeline 1240 includes at least one multithreaded processing unit. The samples output by Vertex Input Buffer 1220 may be processed by any one of the Execution Pipelines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available. 26:62–27:11.</p> <p>Execution Pipelines 1240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 1240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics data stored in buffers in graphics memory via Memory Controller 120. An optional Data Cache 158 within Texture Unit 154 is used to improve memory read performance by reducing read latency. In another alternate embodiment, a Texture Unit 154 is included in each Execution Pipeline 1240. In yet another alternate embodiment, program instructions are stored within Programmable Graphics Processing Pipeline 1250. 27:12–28.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][iii] “to transfer the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit, and”</p>	<p>Kirk discloses “to transfer the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>See 1[c], 1[c][ii].</i></p> <p><i>As a non-limiting example, Kirk discloses the controller (e.g., Graphics Interface 117, Front End 1230, and/or Index Processor 1235) transferring at least a portion of the input data into the accelerator memory (e.g., Local Memory 140) during performance of the intermediate computations in the sequence of computations by at least one graphics processing unit (e.g., Programmable Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250):</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like.</p> <p>3:32–46.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>In various embodiments Memory Controller 120, Local Memory 140, and Geometry Processor 130 are configured such that data generated at various points along Graphics Processing Pipeline 103 may be output via Raster Operation Unit 165 and provided to Geometry Processor 130 or Fragment</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Processor 155 as input. The output data is represented in one or more formats as specified by the codewords. For example, color data may be written as 16, 32, 64, or 128 bits per pixel fixed or floating-point RGBA (red, green, blue, and alpha) to be scanned out for display. As a specific example, four 16-bit floating-point components (RGBA) are combined forming 64 bits of color data for each fragment. The output data, e.g., color, depth, and other parameters, may be processed according to a fragment program and stored in a buffer in graphics memory to be used as a texture map, e.g., shadow map, height field, stencil, and the like, by the fragment program. Alternatively, color and depth output data may be written to a buffer, and later read and processed by Raster Operation Unit 165 to generate the final pixel data prior to being scanned out for display via Output Controller 180.</p> <p>For example, Fragment Processing Pipeline 160 is configured by fragment program instructions to produce processed data and store the processed data in a buffer in Local Memory 140. The Fragment Processing Pipeline 160 is configured by the fragment program instructions to read and further process the processed data. For example, Fragment Processing Pipeline 160 may be configured to implement a modified depth buffer algorithm, e.g., sorting and maintaining more than one depth value for each pixel. A modified depth buffer algorithm may be used to implement correct transparency by rendering fragments in back to front order while applying transparency blending. 5:12–45.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–60:3.</p> <p>Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth,</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:34–55.</p> <p>In step 209 Conflict Detection Unit 152 determines if a RAW position conflict exists for the position associated with the second fragment, and, if so, in step 211 Conflict Detection Unit 152 locks processing of the second fragment. Locking a fragment prevents any processing of the fragment requiring source data that is not yet available due to a RAW position conflict. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory. Step 214 may be completed several, even hundreds of clock cycles after step 205. Raster Operation Unit 165 outputs the write position information to Fragment Processor 155 confirming that the write is complete. In one embodiment the write is considered complete when the write request is output from Memory Controller 120 to Local Memory 140 or to Host Memory 112 via Graphics Interface 117. In another embodiment the write is considered complete when the write request is output from Raster Operation Unit 165 to Memory Controller 120. In step 217 Fragment Processing Pipeline 160 unlocks the second fragment and proceeds to step 220. In step 220 Fragment Processor 155 begins shading the second fragment as specified by the shader. 7:8–29.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 26:56–61.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][iv] “to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit.”</p>	<p>Kirk discloses “to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses transferring at least a portion of output data from accelerator memory (e.g., Local Memory 140) to main memory (e.g., Host Memory 112) during performance of intermediate computations in a sequence of computations by at least one graphics processing unit (e.g., Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250):</i></p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data produced in a pass through Programmable Graphics Processor 105, Graphics Processing Pipeline</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>103 or Fragment Processing Pipeline 160 may be written to a buffer in graphics memory to be read from during a subsequent pass. 4:1–20.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict</p>

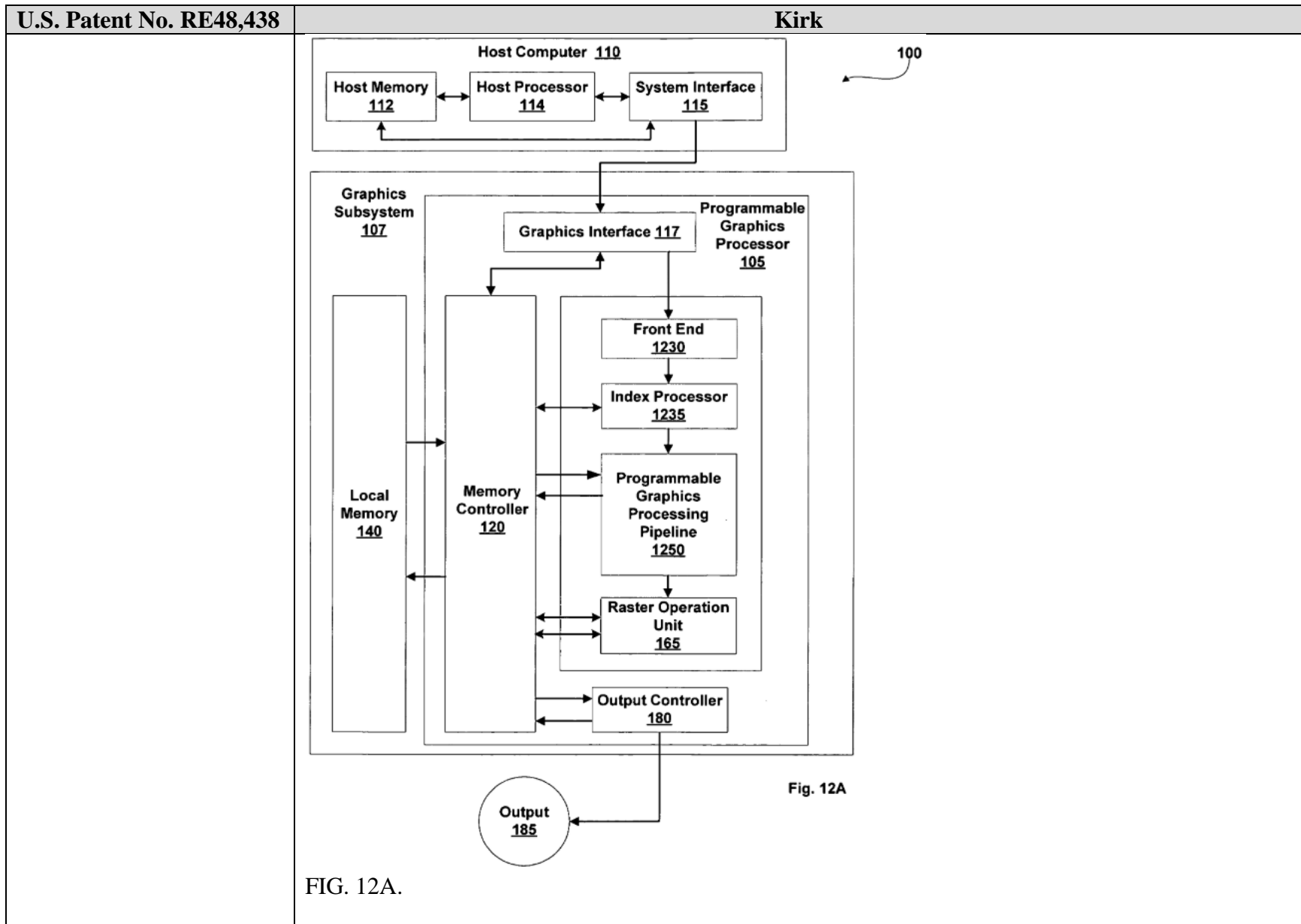
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>2 “The computer system of claim 1, wherein the central processing unit is configured to receive the input data in response to a user interaction.”</p>	<p>Kirk discloses “The computer system of claim 1, wherein the central processing unit is configured to receive the input data in response to a user interaction.” <i>See e.g.:</i></p> <p><i>See claim 1.</i></p> <p><i>As a non-limiting example, Kirk discloses the central processing unit (e.g., Host Processor 114) is configured to receive input data in response to a user interaction (e.g., received through an application programming interface):</i></p> <p>Various embodiments of the invention include an application programming interface for a programmable graphics processor. The application programming interface includes one or more program instruction to configure a fragment processor within the programmable graphics processor to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved. 1:40–47.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs</p>

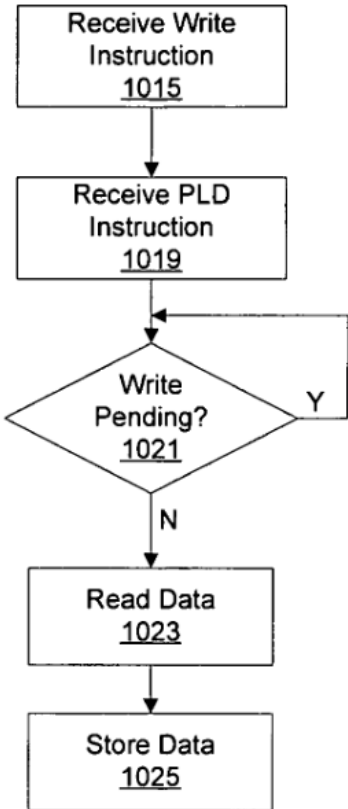
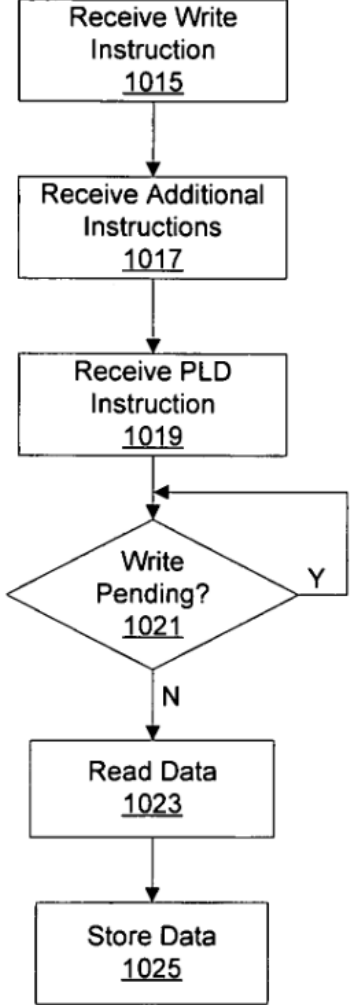
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like.</p> <p>6:4–33.</p> <p>FIGS. 10A, 10B, and 10C illustrate embodiments of methods of processing fragment program instructions, including a PLD (pixel load) instruction in accordance with one or more aspects of the present invention. An API (Application Programming Interface) for a programmable graphics processor includes the PLD instruction to configure Conflict Detection Unit 152 within Programmable Graphics Processor 105 to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved. In some embodiments Conflict Detection Unit 152 is located within Fragment Processor 155.</p> <p>In step 1015, Conflict Detection Unit 152 receives a fragment program instruction specifying a write to a first destination location. The first destination location may be a register in Fragment Processor 155 or a location in graphics memory within a buffer. The first destination location may also include a buffer identification. The buffer may include depth data, color data, stencil data, or the like.</p> <p>In step 1019, Conflict Detection Unit 152 receives a PLD instruction including a source location and a second destination location. In one embodiment the source location is the first destination location and the second destination location is Register 159. In another embodiment the source location is another location within the buffer. In yet another embodiment the source location is another location within another buffer.</p> <p>In step 1021, Conflict Detection Unit 152 determines if a write to the source location is pending, and, if so, Conflict Detection Unit 152 remains in step 1021, waiting until the write to the source location is complete, i.e. for the position conflict to be resolved. Execution of the PLD instruction eliminates the need for executing a flush instruction to drain Fragment Processing Pipeline 160 prior to reading the source location.</p>

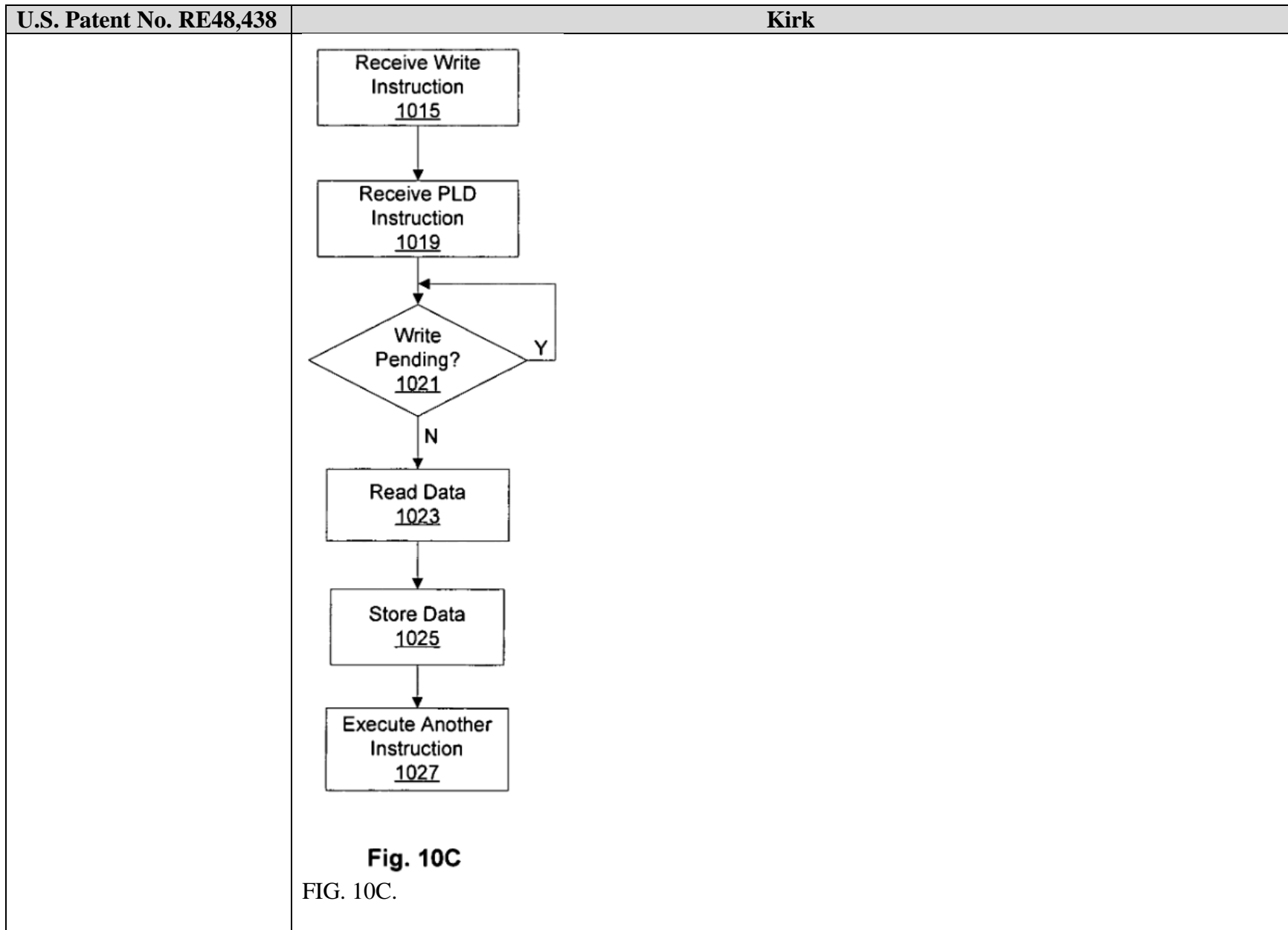
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>If, in step 1021, Conflict Detection Unit 152 determines that a write to the source location is not pending, in step 1023, Read Interface 153 outputs a read request for the source location to Memory Controller 120 and receives the data stored in the source location from Memory Controller 120. In an alternate embodiment Read Interface 153 reads the data stored in the source location from Data Cache 158. In step 1025, Read Interface 153 outputs the data stored in the source location to Fragment Processing Unit 156 and Fragment Processing Unit 156 stores the data in the destination location, e.g. Register 159.</p> <p>FIG. 10B illustrates an embodiment of a method of processing fragment program instructions, including the steps described in relation to FIG. 10A. In step 1015 Conflict Detection Unit 152 receives a fragment program instruction specifying a write to a first destination location. In step 1017 Conflict Detection Unit 152 receives additional fragment program instructions. The additional program instructions may include write instructions specifying other destination locations. Steps 1019, 1021, 1023 and 1025 are completed as described in relation to FIG. 10A.</p> <p>In an embodiment the source location specified in the PLD instruction is the first destination location specified in the fragment program instruction received in step 1015. Execution of the PLD instruction in the embodiment permits reading the source location during processing of the additional fragment program instructions rather than draining Fragment Processor 155 after the write to the first destination location and before executing the source read. Steps 1015, 1019, 1021, 1023, and 1025 are completed as previously described.</p> <p>FIG. 10C illustrates an embodiment of a method of processing fragment program instructions, including the steps described in relation to FIG. 10A. Steps 1015, 1019, 1021, 1023, and 1025 are completed as described in relation to FIG. 10A. In step 1027 Conflict Detection Unit 152 outputs another fragment program instruction to Fragment Processor 155 for execution. In the method illustrated in FIG. 10C, Fragment Processor 155 does not process the other fragment program until the PLD instruction has been executed.</p> <p>18:60–19:67.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>Fig. 10A</p> </div> <div style="text-align: center;">  <p>Fig. 10B</p> </div> </div> <p>FIGs. 10A, 10B.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>3[a] “The computer system of claim 1, wherein the central processing unit is configured to receive the input data at a first rate; and”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein the central processing unit is configured to receive the input data at a first rate.” <i>See e.g.</i>:</p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a Computing System generally designated 100 and including a Host Computer 110 and a Graphics Subsystem 107. Computing System 100 may be a desktop computer, server, laptop computer, palm-sized computer, tablet computer, game console, cellular telephone, computer-based simulator, or the like. Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like.</p> <p>3:17–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>3[b] “the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate.”</p>	<p>Kirk discloses “the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate.”</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group.</p> <p>It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302 producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible.</p> <p>3:12–67.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>Data processed by Geometry Processor 130 and program instructions are passed from Geometry Processor 130 to a Rasterizer 150. Rasterizer 150 is a sampling unit that processes graphics primitives and generates sub-primitive data, such as pixel data or fragment data, including coverage data. Coverage data indicates which sub-pixel sample positions within a pixel are “covered” by a fragment formed by the intersection of the pixel and a primitive. Graphics primitives include geometry, such as points, lines, triangles, quadrilaterals, meshes, surfaces, and the like. Rasterizer 150 converts graphics primitives into sub-primitive data, performing scan conversion on the data processed by Geometry Processor 130. Rasterizer 150 outputs fragment data and fragment program instructions to Fragment Processing Pipeline 160. 4:43–57.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map,</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>FIG. 12B is a block diagram of an exemplary embodiment of Programmable Graphics Processing Pipeline 1250 in accordance with one or more aspects of the present invention. Samples, such as surfaces, primitives, or the like, are received from Index Processor 1235 by Programmable Graphics Processing Pipeline 1250 and stored in a Vertex Input Buffer 1220 in a register file, FIFO (first in first out) memory, cache, or the like (not shown). The samples are broadcast to Execution Pipelines 1240, four of which are shown in FIG. 12B. An alternate embodiment may include either more or fewer Execution Pipelines 1240. Each Execution Pipeline 1240 includes at least one multithreaded processing unit. The samples output by Vertex Input Buffer 1220 may be processed by any one of the Execution Pipelines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available. 26:62–27:11.</p> <p>Execution Pipelines 1240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 1240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics data stored in buffers in graphics memory via Memory Controller 120. An optional Data Cache 158 within Texture Unit 154 is used to improve memory read performance by reducing read latency. In another alternate embodiment, a Texture Unit 154 is included in each Execution Pipeline 1240. In yet another alternate embodiment, program instructions are stored within Programmable Graphics Processing Pipeline 1250.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>27:12–28.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>4 “The computer system of claim 1, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p><i>As a non-limiting example, Kirk discloses main memory (e.g., Host Memory 112) configured to store a copy of output data stored in accelerator memory (e.g., Local Memory 140):</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like.</p> <p>3:32–46.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>5 “The computer system of claim 1, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i].</p> <p>Data 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.” 4:16–20</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Data processed by Geometry Processor 130 and program instructions are passed from Geometry Processor 130 to a Rasterizer 150. Rasterizer 150 is a sampling unit that processes graphics primitives and generates sub-primitive data, such as pixel data or fragment data, including coverage data. Coverage data indicates which sub-pixel sample positions within a pixel are “covered” by a fragment formed by the intersection of the pixel and a primitive. Graphics primitives include geometry, such as points, lines, triangles, quadrilaterals, meshes, surfaces, and the like. Rasterizer 150 converts graphics primitives into sub-primitive data, performing scan conversion on the data processed by Geometry Processor 130. Rasterizer 150 outputs fragment data and fragment program instructions to Fragment Processing Pipeline 160. 4:43–57.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>FIGS. 2A, 2B, and 2C illustrate embodiments of methods of detecting and avoiding position conflicts in accordance with one or more aspects of the present invention. FIG. 2A illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. In step 205 Fragment Processing Pipeline 160 begins shading the first fragment as specified by a fragment program, producing a shaded first fragment, and outputs the shaded first fragment to Raster</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Operation Unit 165. Persons skilled in the art will recognize that depending on the complexity of the fragment program or the depth of a shading pipeline, several clocks cycles, even hundreds of clock cycles may pass before the shaded first fragment is produced. In step 207 Fragment Processing Pipeline 160 receives a second fragment associated with the position within the buffer. To produce a shaded second fragment, the fragment program specifies reading the shaded first fragment as source data. 6:56–7:7.</p> <p>If, in step 209 Conflict Detection Unit 152 determines a RAW position conflict does not exist for the position associated with the second fragment, then in step 220 Fragment Processor 155 begins shading the second fragment as specified by the fragment program producing a shaded second fragment. In step 222 Fragment Processor 155 receives one or more additional fragments, each fragment associated with a position for which a RAW position conflict does not exist. Fragment Processor 155 shades the one or more additional fragments. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory and outputs the write position information to Conflict Detection Unit 152 confirming that the write is complete. 7:30–43.</p> <p>FIG. 2B illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading including the steps illustrated in FIG. 2A. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. The fragment program specifies writing a shaded first fragment to the position within the buffer. In step 203 Conflict Detection Unit 152 receives the position. In one embodiment the position is represented as a pair of coordinates, e.g., (x,y), (s,t), (u,v), and the like, and the coordinates or portions of the coordinates are stored in Conflict Detection Unit 152. The coordinates may be represented relative to a buffer or relative to a display. Coordinates represented within a buffer may be converted into coordinates within a display, e.g., screen coordinates, by applying coordinate offsets based on a position of the buffer within the display. In another embodiment the position is represented as an address for a location in graphics memory. In yet another embodiment the position includes a buffer identifier specifying which of several buffers the position is associated with. In still another embodiment, Conflict Detection Unit 152 identifies a region including the location and stores data, e.g. one or more bits, corresponding to the region. A region may represent several positions, where the</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>positions may correspond to a region of an image, a region of an output buffer, a sequence of physical memory addresses in graphics memory, or the like. Conflict Detection Unit 152 may store data for several regions, depending on a predetermined resolution of the positions to be tracked.</p> <p>In step 205 Fragment Processor 155 begins shading the first fragment, as specified by the fragment program producing a shaded first fragment, several cycles or more later. The shaded first fragment is output to Raster Operation Unit 165. In step 207 Fragment Processing Pipeline 160 receives a second fragment associated with the position within the buffer. To produce a shaded second fragment, the fragment program specifies reading the shaded first fragment as source data. 7:55–8:16.</p> <p>FIG. 4B illustrates a method of processing graphics data including some of the steps shown in FIG. 4A. A fragment program specifies writing data to a location in a buffer to process a first fragment and reading the data from the location in the buffer to produce shaded fragment data for a second fragment without an intervening flush of Fragment Processor 155 or Fragment Processing Pipeline 160.</p> <p>In step 401 fragments are received by Fragment Processing Pipeline 160. In step 402 the location in the buffer to be written by the first fragment is entered in Conflict Detection Unit 152 (CDU). The second fragment is also associated with the location in the buffer, specifically the fragment program specifies using data read from the location (source data) to produce a shaded second fragment. Conflict Detection Unit 152 determines that a write to the location in the buffer is pending and does not initiate reading the location in the buffer. Steps 405, 409, 411, and 413 are completed as previously described in relation to FIG. 4A. 10:39–56.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>Index Processor 1235 optionally reads processed data, e.g., data written by Raster Operation Unit 165, from graphics memory and outputs the data, processed data and formatted commands to Programmable Graphics Processing Pipeline 1250. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each contain one or more programmable processing units to perform a variety of specialized functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 are each optionally configured such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. 26:40–55.</p> <p>In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 26:56–61.</p> <p>FIG. 12B is a block diagram of an exemplary embodiment of Programmable Graphics Processing Pipeline 1250 in accordance with one or more aspects of the present invention. Samples, such as surfaces, primitives, or the like, are received from Index Processor 1235 by Programmable Graphics Processing Pipeline 1250 and stored in a Vertex Input Buffer 1220 in a register file, FIFO (first in first out) memory, cache, or the like (not shown). The samples are broadcast to Execution Pipelines 1240, four of which are shown in FIG. 12B. An alternate embodiment may include either more or fewer Execution Pipelines 1240. Each Execution Pipeline 1240 includes at least one multithreaded processing unit. The samples output by Vertex Input Buffer 1220 may be processed by any one of the Execution Pipelines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>26:62–27:11.</p> <p>Execution Pipelines 1240 may receive first samples, such as higher-order surface data, and tessellate the first samples to generate second samples, such as vertices. Execution Pipelines 1240 may be configured to transform the second samples from an object-based coordinate representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics data stored in buffers in graphics memory via Memory Controller 120. An optional Data Cache 158 within Texture Unit 154 is used to improve memory read performance by reducing read latency. In another alternate embodiment, a Texture Unit 154 is included in each Execution Pipeline 1240. In yet another alternate embodiment, program instructions are stored within Programmable Graphics Processing Pipeline 1250.</p> <p>27:12–28.</p> <p>A graphics program (vertex program or fragment program) is executed within one or more Execution Pipelines 1240 as a plurality of threads where each vertex or fragment to be processed by the program is assigned to a thread. Although threads share processing resources within Programmable Graphics Processing Pipeline 1250 and graphics memory, the execution of each thread proceeds in the one or more Execution Pipelines 1240 independent of any other threads. A RAW position conflict may exist when a fragment program specifies to write to a position in a buffer that the fragment program later specifies to read from. Likewise, a RAW position conflict may exist when a fragment program specifies to write to a position in a buffer that a subsequent fragment program specifies to read from. Furthermore, because threads are executed independently, RAW conflicts may exist when a thread executes a write to a position in a buffer that the thread or another thread executes a read from.</p> <p>27:40–56.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>6[a] “The computer system of claim 1, wherein accelerator memory comprises: a first memory bank to store parameters common to all of the computations in the sequence of computations; and”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein accelerator memory comprises: a first memory bank to store parameters common to all of the computations in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 1.</i></p> <p><i>As a non-limiting example, Kirk discloses a first memory bank (e.g., a logical portion of Local Memory 140) that stores parameters (e.g., graphics data) common to all computations in a sequence of computations:</i></p> <p>Current graphics data processing is exemplified by systems and methods developed to perform a specific operation on several graphics data elements, e.g., linear interpolation, tessellation, texture mapping, depth testing. Traditionally graphics processing systems were implemented as fixed function computation units and more recently the computation units are programmable to perform a limited set of operations. Computation units are connected in a “shading pipeline” to perform shading operations. The shading pipeline includes a texture read interface for reading texture data from graphics memory and a write interface for writing graphics data, including texture data to graphics memory. When the texture data is being written to a buffer stored in graphics memory, the buffer may not be read from by the texture read interface until the shading pipeline has been flushed. The shading pipeline is flushed to assure that any pending writes to the buffer are completed before the texture data is read from the buffer by the shading pipeline. Processing throughput of the shading pipeline is reduced whenever a flush instruction is executed.</p> <p>1:14–33.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105.</p>

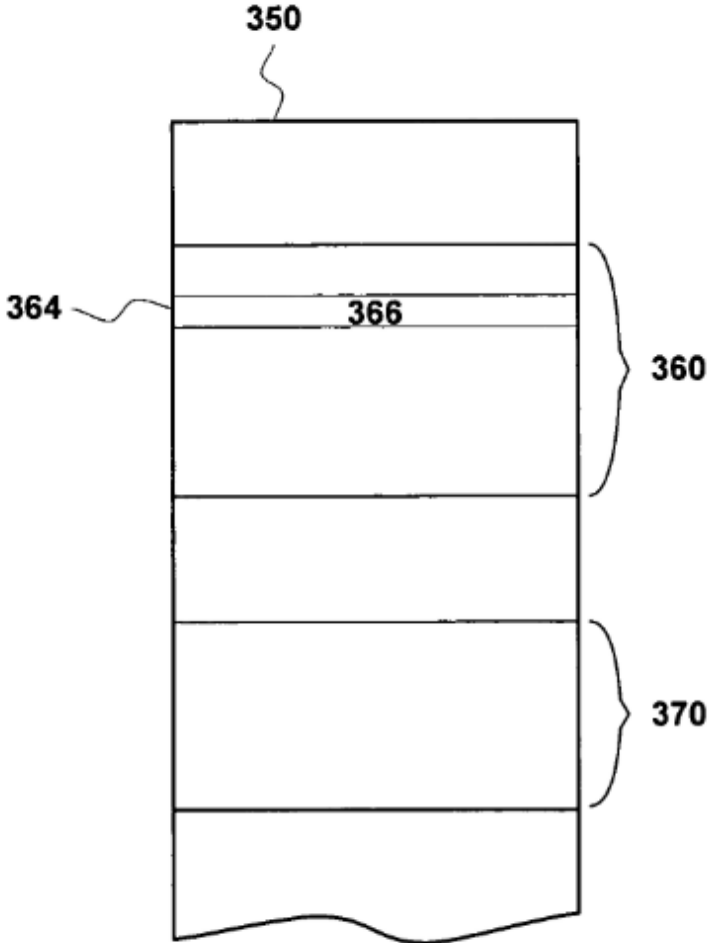
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>4:57–5:11.</p> <p>FIG. 3B illustrates a Portion of Graphics Memory 350 including locations storing data for Buffer 320. Locations within a Section 360 store data for Buffer 320. For example, a Location 366 stores data associated with Pixel 340, e.g., color, depth, stencil, shadow depth, and the like. An Address 364 is used to access the Location 366. Address 364 may be computed based on an x,y position and a base address corresponding to a first location within Section 360. In an alternate embodiment Address 364 is computed based on a position within Buffer 320 and an address offset within Portion of Graphics Memory 350 corresponding to Section 360. A Section 370 includes locations storing data for another buffer. Each buffer is associated with a unique buffer identifier that may be used to determine a corresponding base address.</p> <p>9:41–55.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	 <p data-bbox="926 1222 1041 1260">Fig. 3B</p> <p data-bbox="583 1284 688 1317">Fig. 3B.</p> <p data-bbox="583 1357 1881 1425">To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>6[b] “a second memory bank to store data specific to at least one computation in the sequence of computations.”</p>	<p>Kirk discloses “a second memory bank to store data specific to at least one computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p><i>As a non-limiting example, Kirk discloses a second memory bank (e.g., a logical portion of Local Memory 140) that stores data (e.g., data input to the computations) specific to at least one computation in a sequence of computations:</i></p> <p>Current graphics data processing is exemplified by systems and methods developed to perform a specific operation on several graphics data elements, e.g., linear interpolation, tessellation, texture mapping, depth testing. Traditionally graphics processing systems were implemented as fixed function computation units and more recently the computation units are programmable to perform a limited set of operations. Computation units are connected in a “shading pipeline” to perform shading operations. The shading pipeline includes a texture read interface for reading texture data from graphics memory and a write interface for writing graphics data, including texture data to graphics memory. When the texture data is being written to a buffer stored in graphics memory, the buffer may not be read from by the texture read interface until the shading pipeline has been flushed. The shading pipeline is flushed to assure that any pending writes to the buffer are completed before the texture data is read from the buffer by the shading pipeline. Processing throughput of the shading pipeline is reduced whenever a flush instruction is executed.</p> <p>1:14–33.</p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data</p>

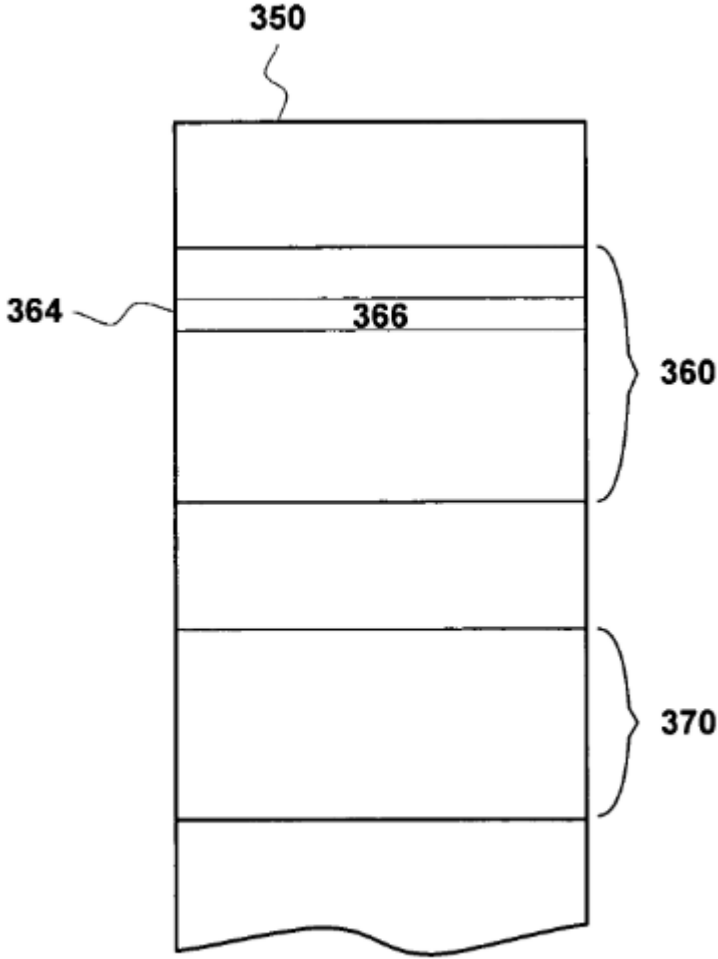
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data produced in a pass through Programmable Graphics Processor 105, Graphics Processing Pipeline</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>103 or Fragment Processing Pipeline 160 may be written to a buffer in graphics memory to be read from during a subsequent pass. 4:1–20.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>FIG. 3B illustrates a Portion of Graphics Memory 350 including locations storing data for Buffer 320. Locations within a Section 360 store data for Buffer 320. For example, a Location 366 stores data associated with Pixel 340, e.g., color, depth, stencil, shadow depth, and the like. An Address 364 is used to access the Location 366. Address 364 may be computed based on an x,y position and a base address corresponding to a first location within Section 360. In an alternate embodiment Address 364 is computed based on a position within Buffer 320 and an address offset within Portion of Graphics Memory 350 corresponding to Section 360. A Section 370 includes locations storing data for another buffer. Each buffer is associated with a unique buffer identifier that may be used to determine a corresponding base address. 9:41–55.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	 <p data-bbox="932 1235 1052 1273">Fig. 3B</p> <p data-bbox="583 1300 695 1338">Fig. 3B.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>7 “The computer system of claim 1, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.” <i>See e.g.:</i></p> <p><i>See 1[d][iv].</i></p> <p><i>As a non-limiting example, Kirk discloses the controller (e.g., Graphics Interface 117, Front End 1230, Index Processor 1235, and/or Memory Controller 120) is configured to transfer the output data from accelerator memory (e.g., Local Memory 140) to main memory (e.g., Host Memory 112) without transferring intermediate results:</i></p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235,</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>8 “The computer system of claim 1, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations.”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p><i>As a non-limiting example, Kirk discloses the controller (e.g., Graphics Interface 117, Front End 1230, Index Processor 1235, and/or Memory Controller 120) is configured to transfer output data from accelerator memory (e.g., Local Memory 140) to main memory (e.g., Host Memory 112) after the graphics processing unit (e.g., Programmable Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250) begins another sequence of computations:</i></p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory</p>

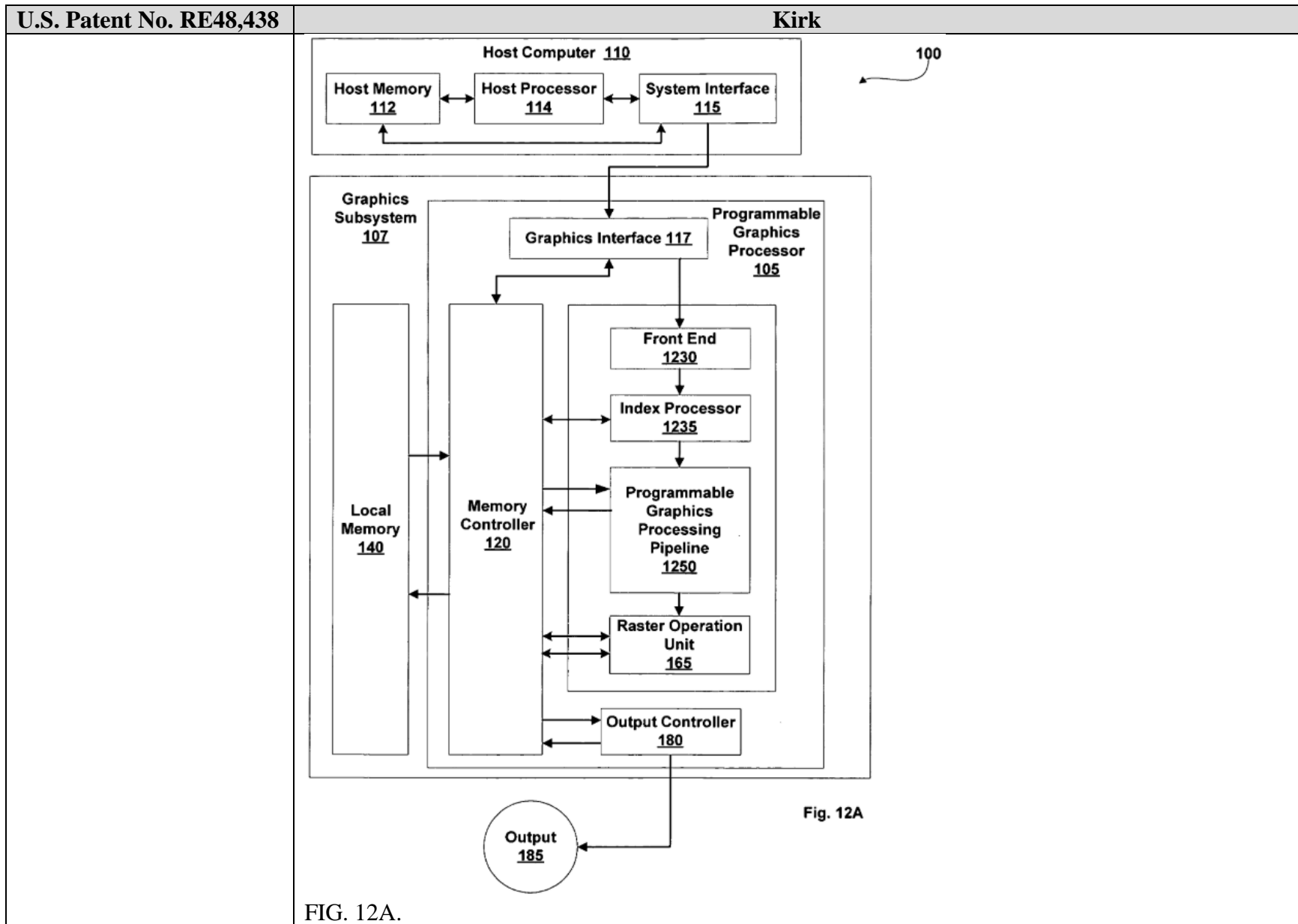
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes,</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>9 “The computer system of claim 8, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit.”</p>	<p>Kirk discloses “[t]he computer system of claim 8, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii], 1[d][iv]; claim 8.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>10 “The computer system of claim 1, wherein the controller is configured to control execution of the sequence of computations by the at least one graphics processing unit.”</p>	<p>Kirk discloses “[t]he computer system of claim 1, wherein the controller is configured to control execution of the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][ii].</p> <p><i>As a non-limiting example, Kirk discloses the controller (e.g., Graphics Interface 117, Front End 1230, Index Processor 1235, and/or Memory Controller 120) controls execution of computations by</i></p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p><i>the graphics processing unit (e.g., Programmable Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250):</i></p> <p>FIG. 1A is a block diagram of an exemplary embodiment of a Computing System generally designated 100 and including a Host Computer 110 and a Graphics Subsystem 107. Computing System 100 may be a desktop computer, server, laptop computer, palm-sized computer, tablet computer, game console, cellular telephone, computer-based simulator, or the like. Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:17–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[pre] “A method of performing a sequence of computations representing an artificial neural network on a computer system comprising a central processing unit (CPU), a main memory operably coupled to the central processing unit via a bus, an accelerator operably coupled to the CPU and the</p>	<p>To the extent the preamble is limiting, Kirk discloses “[a] method of performing a sequence of computations representing an artificial neural network on a computer system comprising a central processing unit (CPU), a main memory operably coupled to the central processing unit via a bus, an accelerator operably coupled to the CPU and the main memory via the bus, the accelerator comprising a graphics processing unit (GPU) and an accelerator memory.”</p> <p><i>See</i> 1[a]-1[c][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>main memory via the bus, the accelerator comprising a graphics processing unit (GPU) and an accelerator memory, the method comprising:”</p>	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[a] “(A) performing, by the GPU, the sequence of computations on a first portion of input data so as to generate a first portion of output data, the first portion of the output data representing an output of a neuron in a first layer of the artificial neural network, intermediate computations in the sequence of computations yielding intermediate results, wherein performing the sequence of computations on the first portion of the input data comprises”</p>	<p>Kirk discloses “performing, by the GPU, the sequence of computations on a first portion of input data so as to generate a first portion of output data, the first portion of the output data representing an output of a neuron in a first layer of the artificial neural network, intermediate computations in the sequence of computations yielding intermediate results.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[a][i] “(i) assigning an output variable to a first texture and a second texture, the output variable being included in a first computational element of a plurality of computational</p>	<p>Kirk discloses “assigning an output variable to a first texture and a second texture, the output variable being included in a first computational element of a plurality of computational elements, the plurality of computational elements representing the sequence of computations.” <i>See e.g.:</i></p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>elements, the plurality of computational elements representing the sequence of computations and”</p>	<p>embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[a][ii] “(ii) accumulating a first value for the output variable in the first texture during a first time step;”</p>	<p>Kirk discloses “accumulating a first value for the output variable in the first texture during a first time step.” <i>See e.g.</i>:</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>11:22–41.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[b] “(B) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the input data from the main memory to the accelerator via the bus;”</p>	<p>Kirk discloses “in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the input data from the main memory to the accelerator via the bus.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii]; claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[c] “(C) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus,</p>	<p>Kirk discloses “in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus, the second portion of the output data representing an output of a neuron in a second layer in the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv]; claim 8.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>the second portion of the output data representing an output of a neuron in a second layer in the artificial neural network; and”</p>	<p><i>As a non-limiting example, Kirk discloses transferring outputs of computations from accelerator memory (e.g., Local Memory 140) to main memory (e.g., Host Memory 112):</i></p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data 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. 4:1–20.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p>

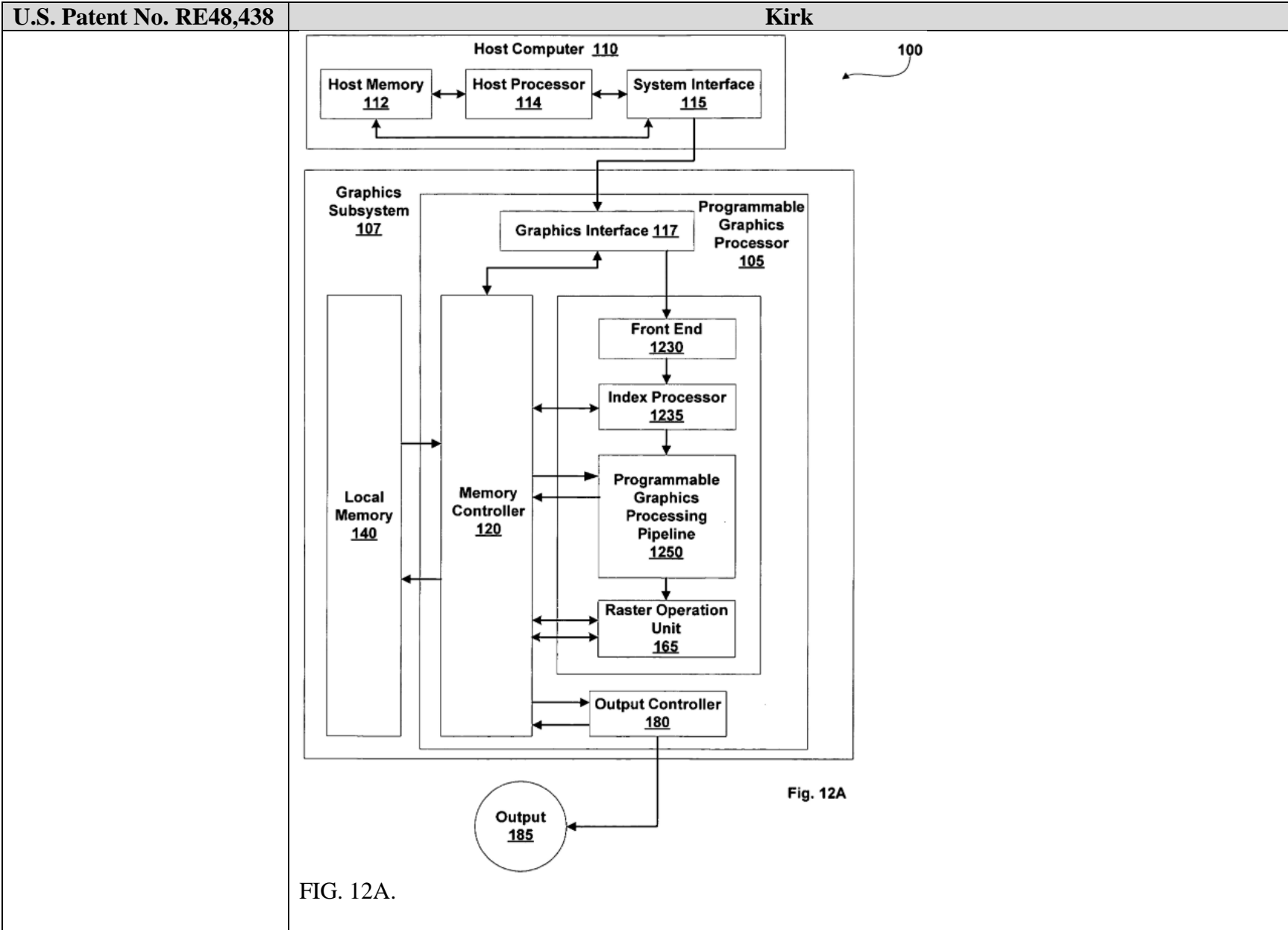
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise,</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[d] “(D) performing, by the GPU, the sequence of computations on the second portion of the input data, wherein performing the sequence of computations on the second portion of the input data comprises”</p>	<p>Kirk discloses “performing, by the GPU, the sequence of computations on the second portion of the input data.” <i>See e.g.:</i></p> <p><i>See</i> 12[a], 12[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[d][i] “(i) accumulating a second value for the output variable in the second texture during a second time step and”</p>	<p>Kirk discloses “accumulating a second value for the output variable in the second texture during a second time step.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 12[a][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[d][ii] “(ii) making the first value of the output variable in the first texture accessible to other computational elements in the plurality of computational elements during the second time step.”</p>	<p>Kirk discloses “making the first value of the output variable in the first texture accessible to other computational elements in the plurality of computational elements during the second time step.” <i>See e.g.:</i></p> <p><i>See</i> 12[d][i], 1[c][ii].</p> <p><i>As a non-limiting example, Kirk discloses making the first value of the output variable in the first texture accessible to other computational elements during the second time step (e.g., by writing data produced by the processing pipeline to a buffer in graphics memory to be read from during a subsequent pass):</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like.</p> <p>3:32–46.</p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data 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. 4:1–20.</p> <p>In various embodiments Memory Controller 120, Local Memory 140, and Geometry Processor 130 are configured such that data generated at various points along Graphics Processing Pipeline 103 may be output via Raster Operation Unit 165 and provided to Geometry Processor 130 or Fragment Processor 155 as input. The output data is represented in one or more formats as specified by the codewords. For example, color data may be written as 16, 32, 64, or 128 bits per pixel fixed or floating-point RGBA (red, green, blue, and alpha) to be scanned out for display. As a specific example, four 16-bit floating-point components (RGBA) are combined forming 64 bits of color data for each fragment. The output data, e.g., color, depth, and other parameters, may be processed according to a fragment program and stored in a buffer in graphics memory to be used as a texture map, e.g., shadow map, height field, stencil, and the like, by the fragment program. Alternatively, color and depth output data may be written to a buffer, and later read and processed by Raster Operation Unit 165 to generate the final pixel data prior to being scanned out for display via Output Controller 180.</p> <p>For example, Fragment Processing Pipeline 160 is configured by fragment program instructions to produce processed data and store the processed data in a buffer in Local Memory 140. The Fragment Processing Pipeline 160 is configured by the fragment program instructions to read and further process the processed data. For example, Fragment Processing Pipeline 160 may be configured to implement a modified depth buffer algorithm, e.g., sorting and maintaining more than one depth value for each pixel. A modified depth buffer algorithm may be used to implement correct transparency by rendering fragments in back to front order while applying transparency blending. 5:12–45.</p> <p>A graphics program (vertex program or fragment program) is executed within one or more Execution Pipelines 1240 as a plurality of threads where each vertex or fragment to be processed by the program is assigned to a thread. Although threads share processing resources within Programmable Graphics Processing Pipeline 1250 and graphics memory, the execution of each thread proceeds in the one or</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>more Execution Pipelines 1240 independent of any other threads. A RAW position conflict may exist when a fragment program specifies to write to a position in a buffer that the fragment program later specifies to read from. Likewise, a RAW position conflict may exist when a fragment program specifies to write to a position in a buffer that a subsequent fragment program specifies to read from. Furthermore, because threads are executed independently, RAW conflicts may exist when a thread executes a write to a position in a buffer that the thread or another thread executes a read from. 27:40–56.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>13 “The method of claim 12, further comprising: storing the input data in the main memory in response to a user interaction.”</p>	<p>Kirk discloses “[t]he method of claim 12, further comprising: storing the input data in the main memory in response to a user interaction.” <i>See e.g.</i>:</p> <p><i>See</i> 1[b]; claim 2.</p> <p><i>As a non-limiting example, Kirk discloses storing input data in main memory (e.g., Host Memory 112) in response to a user interaction received through an API:</i></p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4-55.</p> <p>FIGS. 10A, 10B, and 10C illustrate embodiments of methods of processing fragment program instructions, including a PLD (pixel load) instruction in accordance with one or more aspects of the present invention. An API (Application Programming Interface) for a programmable graphics processor includes the PLD instruction to configure Conflict Detection Unit 152 within Programmable Graphics Processor 105 to detect a position conflict for a position and prevent a subsequent access of the position until the position conflict is resolved. In some embodiments Conflict Detection Unit 152 is located within Fragment Processor 155.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>In step 1015, Conflict Detection Unit 152 receives a fragment program instruction specifying a write to a first destination location. The first destination location may be a register in Fragment Processor 155 or a location in graphics memory within a buffer. The first destination location may also include a buffer identification. The buffer may include depth data, color data, stencil data, or the like.</p> <p>In step 1019, Conflict Detection Unit 152 receives a PLD instruction including a source location and a second destination location. In one embodiment the source location is the first destination location and the second destination location is Register 159. In another embodiment the source location is another location within the buffer. In yet another embodiment the source location is another location within another buffer.</p> <p>In step 1021, Conflict Detection Unit 152 determines if a write to the source location is pending, and, if so, Conflict Detection Unit 152 remains in step 1021, waiting until the write to the source location is complete, i.e. for the position conflict to be resolved. Execution of the PLD instruction eliminates the need for executing a flush instruction to drain Fragment Processing Pipeline 160 prior to reading the source location.</p> <p>If, in step 1021, Conflict Detection Unit 152 determines that a write to the source location is not pending, in step 1023, Read Interface 153 outputs a read request for the source location to Memory Controller 120 and receives the data stored in the source location from Memory Controller 120. In an alternate embodiment Read Interface 153 reads the data stored in the source location from Data Cache 158. In step 1025, Read Interface 153 outputs the data stored in the source location to Fragment Processing Unit 156 and Fragment Processing Unit 156 stores the data in the destination location, e.g. Register 159.</p> <p>FIG. 10B illustrates an embodiment of a method of processing fragment program instructions, including the steps described in relation to FIG. 10A. In step 1015 Conflict Detection Unit 152 receives a fragment program instruction specifying a write to a first destination location. In step 1017 Conflict Detection Unit 152 receives additional fragment program instructions. The additional program instructions may include write instructions specifying other destination locations. Steps 1019, 1021, 1023 and 1025 are completed as described in relation to FIG. 10A.</p> <p>In an embodiment the source location specified in the PLD instruction is the first destination location specified in the fragment program instruction received in step 1015. Execution of the PLD instruction in the embodiment permits reading the source location during processing of the additional fragment program instructions rather than draining Fragment Processor 155 after the write to the first</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>destination location and before executing the source read. Steps 1015, 1019, 1021, 1023, and 1025 are completed as previously described.</p> <p>FIG. 10C illustrates an embodiment of a method of processing fragment program instructions, including the steps described in relation to FIG. 10A. Steps 1015, 1019, 1021, 1023, and 1025 are completed as described in relation to FIG. 10A. In step 1027 Conflict Detection Unit 152 outputs another fragment program instruction to Fragment Processor 155 for execution. In the method illustrated in FIG. 10C, Fragment Processor 155 does not process the other fragment program until the PLD instruction has been executed.</p> <p>18:60–19:67.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>14[a] “The method of claim 12, further comprising: receiving the input data at a first rate; and”</p>	<p><i>See</i> 3[a]; claim 12.</p>
<p>14[b] “wherein (A) comprises performing the sequence of computations at a second rate different than the first rate.”</p>	<p><i>See</i> 3[b].</p>
<p>16 “The method of claim 12, wherein (C) comprises: transferring the second portion of the output data</p>	<p>Kirk discloses “[t] he method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus.” <i>See e.g.</i>:</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus.”</p>	<p><i>See claim 7.</i></p> <p><i>As a non-limiting example, Kirk discloses transferring a second portion of output data from accelerator memory (e.g., Local Memory 140) to main memory (e.g., Host Memory 112) without transferring intermediate results:</i></p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152</p>

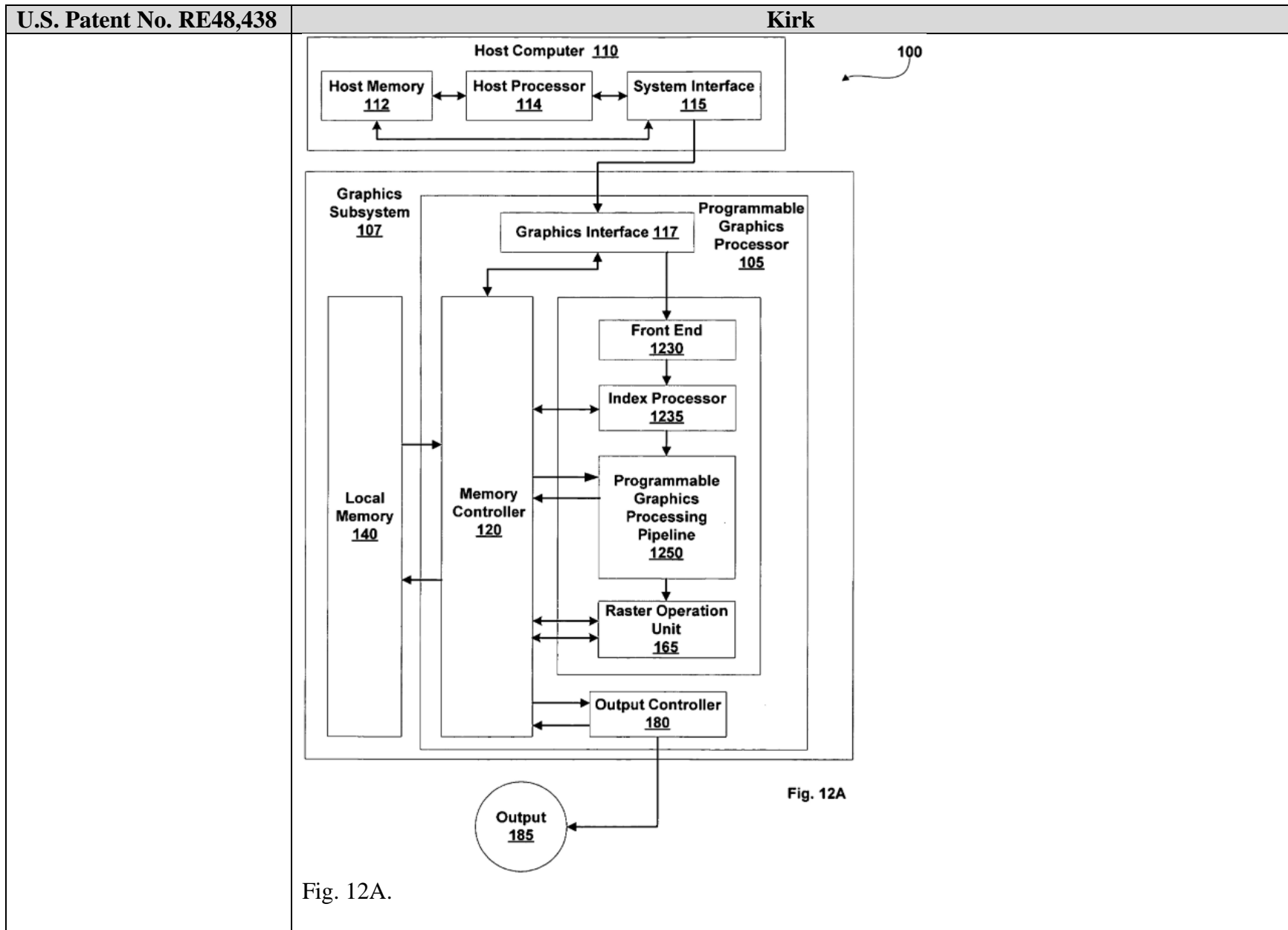
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4-55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>17 “The method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory after the GPU has begun to perform another sequence of computations.”</p>	<p>Kirk discloses “[t]he method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory after the GPU has begun to perform another sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 8.</i></p> <p><i>As a non-limiting example, Kirk discloses transferring a second portion of output data from accelerator memory (e.g., Local Memory 140) to main memory (e.g., Host Memory 112) after the graphics processing unit (e.g., Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250) begins another sequence of computations:</i></p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data.</p>

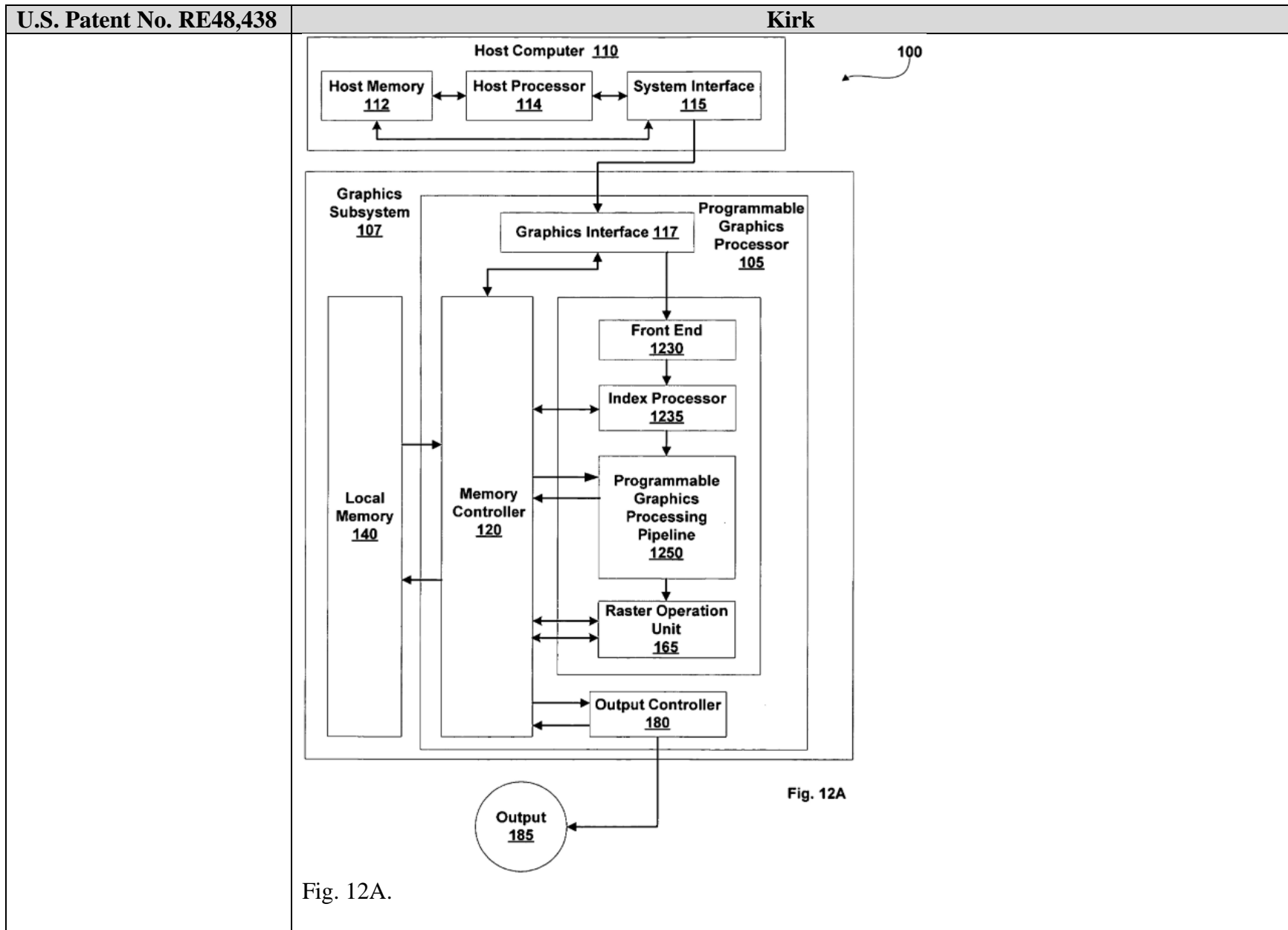
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>4:57–5:11.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like.</p> <p>5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth,</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>Texture Unit 154 outputs the texture map data, processed fragment data, and the additional program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 stores the buffer data in a Register 159 to be used as source data. Fragment Processing Unit 156 processes the processed map data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally-processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts. Write Interface 157 outputs the position information and processed shaded fragment data to Data Cache 158 to update the entry. 11:22–41.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk



Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>18 “The method of claim 17, wherein (C) further comprises: initiating transfer of the second portion of the output data in parallel with performance of at least one computation in the other sequence of computations.”</p>	<p>Kirk discloses “[t]he method of claim 17, wherein (C) further comprises: initiating transfer of the second portion of the output data in parallel with performance of at least one computation in the other sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>20[a] “The method of claim 12, further comprising: storing parameters common to all of the computations in the sequence of computations in a first memory bank in the accelerator memory; and”</p>	<p>Kirk discloses “[t]he method of claim 12, further comprising: storing parameters common to all of the computations in the sequence of computations in a first memory bank in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 6[a]; claim 12.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>20[b] “storing data specific to at least one computation in the sequence of computations in a second memory bank in the accelerator memory.”</p>	<p>Kirk discloses “storing data specific to at least one computation in the sequence of computations in a second memory bank in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 6[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[pre] “A method of performing a sequence of computations representing an artificial neural network, the method comprising:”</p>	<p>To the extent the preamble is limiting, Kirk discloses “[a] method of performing a sequence of computations representing an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[pre].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[a] “receiving, at a central processing unit</p>	<p>Kirk discloses “receiving, at a central processing unit (CPU), first input data acquired from an external system in real time.” <i>See e.g.:</i></p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>(CPU), first input data acquired from an external system in real time;”</p>	<p><i>See</i> 1[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[b] “initializing, by a controller operably coupled to a graphics processing unit (GPU), textures and shaders in a memory operably coupled to the GPU;”</p>	<p>Kirk discloses “initializing, by a controller operably coupled to a graphics processing unit (GPU), textures and shaders in a memory operably coupled to the GPU.” <i>See e.g.</i>:</p> <p><i>See</i> 1[c][ii], 1[d], 1[d][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[c] “transferring the first input data received by the CPU to the memory operably coupled to the GPU;”</p>	<p>Kirk discloses “transferring the first input data received by the CPU to the memory operably coupled to the GPU.” <i>See e.g.</i>:</p> <p><i>See</i> 1[b], 1[c], 1[d][iii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[d] “performing, by the graphics processing unit (GPU), a first computation in the sequence of computations on the first input data based on the textures and shaders to generate first output data, computations in the sequence of computations representing respective layers of neurons in the artificial neural network, an output of the first computation in the sequence of computations representing an output of a first neuron in a first layer in the artificial neural network;”</p>	<p>Kirk discloses “performing, by the graphics processing unit (GPU), a first computation in the sequence of computations on the first input data based on the textures and shaders to generate first output data, computations in the sequence of computations representing respective layers of neurons in the artificial neural network, an output of the first computation in the sequence of computations representing an output of a first neuron in a first layer in the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[e] “storing, in the memory operably coupled to the GPU, the first input data and the first output data; and”</p>	<p>Kirk discloses “storing, in the memory operably coupled to the GPU, the first input data and the first output data.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][ii], 1[d][iii], 12[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[f] “transferring second input data acquired from the external system in real time into the memory operably coupled to the GPU after the GPU starts the first computation and before the GPU starts a second computation of the sequence of computations, an output of the second computation in the sequence of computations representing an output of a second neuron in a second layer in the artificial neural network.”</p>	<p>Kirk discloses “transferring second input data acquired from the external system in real time into the memory operably coupled to the GPU after the GPU starts the first computation and before the GPU starts a second computation of the sequence of computations, an output of the second computation in the sequence of computations representing an output of a second neuron in a second layer in the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii], 1[d][iv], 12[b], 12[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>22 “The method of claim 21, wherein transferring the second input data comprises transferring the second input data via a bus operably coupled to the CPU.”</p>	<p>Kirk discloses “[t]he method of claim 21, wherein transferring the second input data comprises transferring the second input data via a bus operably coupled to the CPU.” <i>See e.g.:</i></p> <p><i>See</i> 1[c], 12[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>23 “The method of claim 21, further comprising: transferring the first output data from the memory to another memory during the second computation in the sequence of computations.”</p>	<p>Kirk discloses “[t]he method of claim 21, further comprising: transferring the first output data from the memory to another memory during the second computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p><i>As a non-limiting example, Kirk discloses transferring output data from one memory (e.g., local register file 304, global register file 306, or memory 308) to another memory (e.g., a different memory in the accelerator or system memory 104).</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>24[a] “The method of claim 23, further comprising: storing intermediate results of the sequence of computations in the memory, and”</p>	<p>Kirk discloses “[t]he method of claim 23, further comprising: storing intermediate results of the sequence of computations in the memory.” <i>See e.g.:</i></p> <p><i>See</i> 12[a]; claim 16.</p> <p><i>As a non-limiting example, Kirk discloses storing intermediate results in memory (e.g., Local Memory 140 and/or Host Memory 112):</i></p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>When processing is completed, an Output 185 of Graphics Subsystem 107 is provided using Output Controller 180. Alternatively, Host Processor 114 reads the composited frame, e.g., buffer, stored in Local Memory 140 through Memory Controller 120, Graphics Interface 117 and System Interface 115. Output Controller 180 is optionally configured by opcodes, received from Graphics Processing Pipeline 103 via Memory Controller 120, to deliver data to a display device, network, electronic control system, other Computing System 100, other Graphics Subsystem 110, or the like. 5:60–6:3.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein. 6:4–55.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
24[b] “wherein transferring the first output data from the memory to the other	Kirk discloses “wherein transferring the first output data from the memory to the other memory occurs without transferring the intermediate results of the sequence of computations.” <i>See e.g.:</i>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>memory occurs without transferring the intermediate results of the sequence of computations.”</p>	<p><i>See</i> claims 7, 16.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>25 “The method of claim 23, wherein transferring the second input data and transferring the first output data occurs in parallel.”</p>	<p>Kirk discloses “[t]he method of claim 23, wherein transferring the second input data and transferring the first output data occurs in parallel.”</p> <p><i>See</i> 12[b], 12[c]; claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>26 “The method of claim 21, further comprising: storing, in a first memory partition of the memory, parameters common to all of the computations in the sequence of computations.”</p>	<p>Kirk discloses “[t]he method of claim 21, further comprising: storing, in a first memory partition of the memory, parameters common to all of the computations in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 6[a], claim 21.</p> <p><i>As a non-limiting example, Kirk discloses a first memory partition of the memory (e.g., a logical portion of Local Memory 140) that stores parameters (e.g., graphics data) common to all computations in a sequence of computations:</i></p>

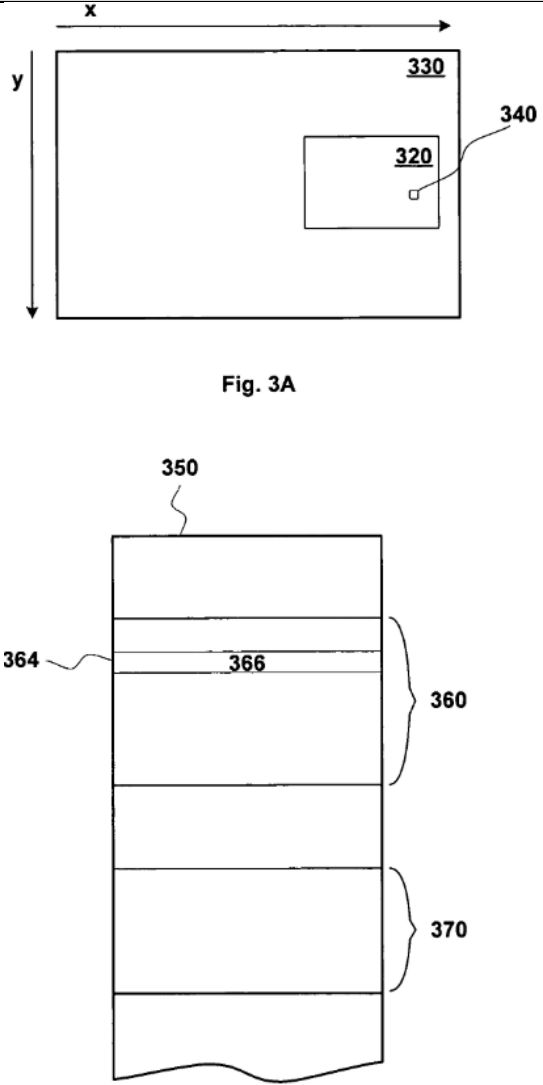
Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>FIG. 2B illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading including the steps illustrated in FIG. 2A. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. The fragment program specifies writing a shaded first fragment to the position within the buffer. In step 203 Conflict Detection Unit 152 receives the position. In one embodiment the position is represented as a pair of coordinates, e.g., (x,y), (s,t), (u,v), and the like, and the coordinates or portions of the coordinates are stored in Conflict Detection Unit 152. The coordinates may be represented relative to a buffer or relative to a display. Coordinates represented within a buffer may be converted into coordinates within a display, e.g., screen coordinates, by applying coordinate offsets based on a position of the buffer within the display. In another embodiment the position is represented as an address for a location in graphics memory. In yet another embodiment the position includes a buffer identifier specifying which of several buffers the position is associated with. In still another embodiment, Conflict Detection Unit 152 identifies a region including the location and stores data, e.g. one or more bits, corresponding to the region. A region may represent several positions, where the positions may correspond to a region of an image, a region of an output buffer, a sequence of physical</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>memory addresses in graphics memory, or the like. Conflict Detection Unit 152 may store data for several regions, depending on a predetermined resolution of the positions to be tracked. In step 205 Fragment Processor 155 begins shading the first fragment, as specified by the fragment program producing a shaded first fragment, several cycles or more later. The shaded first fragment is output to Raster Operation Unit 165. In step 207 Fragment Processing Pipeline 160 receives a second fragment associated with the position within the buffer. To produce a shaded second fragment, the fragment program specifies reading the shaded first fragment as source data. 7:55–8:16.</p> <p>FIG. 3A is a conceptual diagram of a Buffer 320 displayed by a display device, e.g., monitor, projector, and the like. Data stored in Buffer 320 is displayed on Display 330. Additional buffers of arbitrary sizes may be displayed on Display 330. Each buffer may be positioned for display relative to Display 330. A Pixel 340 within Buffer 320 is associated with an x,y position relative to Display 330. In an alternate embodiment the x,y origin is in the lower left corner of Display 330. 9:32–55.</p> <p>FIG. 3B illustrates a Portion of Graphics Memory 350 including locations storing data for Buffer 320. Locations within a Section 360 store data for Buffer 320. For example, a Location 366 stores data associated with Pixel 340, e.g., color, depth, stencil, shadow depth, and the like. An Address 364 is used to access the Location 366. Address 364 may be computed based on an x,y position and a base address corresponding to a first location within Section 360. In an alternate embodiment Address 364 is computed based on a position within Buffer 320 and an address offset within Portion of Graphics Memory 350 corresponding to Section 360. A Section 370 includes locations storing data for another buffer. Each buffer is associated with a unique buffer identifier that may be used to determine a corresponding base address. 9:41–55.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	 <p data-bbox="800 597 877 621">Fig. 3A</p> <p data-bbox="800 1336 877 1360">Fig. 3B</p> <p data-bbox="583 1373 758 1398">Figs. 3A, 3B.</p> <p>The diagram shows two figures. Fig. 3A is a rectangular frame with a horizontal dimension 'x' and a vertical dimension 'y'. Inside the frame is a smaller rectangle labeled '320'. A line labeled '340' points to the bottom-right corner of rectangle '320'. The label '330' is in the top-right corner of the outer frame. Fig. 3B is a vertical stack of five rectangular layers. The top layer is labeled '350'. The second layer from the top is labeled '366'. A bracket on the right side of the second and third layers is labeled '360'. A bracket on the right side of the fourth and fifth layers is labeled '370'. A line labeled '364' points to the left edge of the second layer.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>27 “The method of claim 26, further comprising: storing, in a second memory partition of the memory, data specific to the first computation in the sequence of computations.”</p>	<p>Kirk discloses “[t]he method of claim 26, further comprising: storing, in a second memory partition of the memory, data specific to the first computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 6[a], 6[b]; claim 26.</p> <p><i>As a non-limiting example, Kirk discloses a second memory partition of the memory (e.g., a logical portion of Local Memory 140 and/or Host Memory 112) that stores data (e.g., graphics data) specific to at least one computation in a sequence of computations.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>28 “The method of claim 27, further comprising: storing, in the second memory partition, external input data patterns, representations of internal</p>	<p>Kirk discloses “[t]he method of claim 27, further comprising: storing, in the second memory partition, external input data patterns, representations of internal variables, an input of the computation in the sequence of computations, and the output of the computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> claim 27.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>variables, an input of the computation in the sequence of computations, and the output of the computation in the sequence of computations.”</p>	<p><i>As a non-limiting example, Kirk discloses storing external input data patterns (e.g., input from an external device) and representations of internal variables:</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data 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. 4:1–20.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p><i>Kirk further discloses storing inputs and outputs of computations:</i></p> <p><i>See 1[c][ii], 1[d][iii].</i></p> <p><i>Kirk discloses storing each type of data in a second “memory partition”:</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>The fragment programs configure the Fragment Processing Pipeline 160 to process fragment data by specifying computations and computation precision. A Fragment Processor 155 optionally is configured by fragment program instructions such that fragment data processing operations are</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>performed in multiple internal passes within Fragment Processor 155. Fragment Processor 155 outputs processed fragment data and codewords generated from fragment program instructions to Raster Operation Unit 165. Raster Operation Unit 165 includes a read interface and a write interface to Memory Controller 120 through which Raster Operation Unit 165 accesses data stored in one or more buffers in Local Memory 140 or Host Memory 112. Raster Operation Unit 165 optionally performs near and far plane clipping and raster operations, such as stencil, z test, blending, and the like, using fragment data read from the one or more buffers in Local Memory 140 or Host Memory 112 at the x,y position associated with the fragment data and the processed fragment data to produce output data. The output data from Raster Operation Unit 165 is written back to Local Memory 140 or Host Memory 112 at the x,y position associated with the output data. 4:57–5:11.</p> <p>In various embodiments Memory Controller 120, Local Memory 140, and Geometry Processor 130 are configured such that data generated at various points along Graphics Processing Pipeline 103 may be output via Raster Operation Unit 165 and provided to Geometry Processor 130 or Fragment Processor 155 as input. The output data is represented in one or more formats as specified by the codewords. For example, color data may be written as 16, 32, 64, or 128 bits per pixel fixed or floating-point RGBA (red, green, blue, and alpha) to be scanned out for display. As a specific example, four 16-bit floating-point components (RGBA) are combined forming 64 bits of color data for each fragment. The output data, e.g., color, depth, and other parameters, may be processed according to a fragment program and stored in a buffer in graphics memory to be used as a texture map, e.g., shadow map, height field, stencil, and the like, by the fragment program. Alternatively, color and depth output data may be written to a buffer, and later read and processed by Raster Operation Unit 165 to generate the final pixel data prior to being scanned out for display via Output Controller 180.</p> <p>For example, Fragment Processing Pipeline 160 is configured by fragment program instructions to produce processed data and store the processed data in a buffer in Local Memory 140. The Fragment Processing Pipeline 160 is configured by the fragment program instructions to read and further process the processed data. For example, Fragment Processing Pipeline 160 may be configured to implement a modified depth buffer algorithm, e.g., sorting and maintaining more than one depth value for each pixel. A modified depth buffer algorithm may be used to implement correct transparency by rendering fragments in back to front order while applying transparency blending.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>5:12–45.</p> <p>FIG. 1B is a block diagram of an exemplary embodiment of portions of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention. A Conflict Detection Unit 152 receives fragment data and fragment program instructions from Rasterizer 150. In an alternate embodiment, Conflict Detection Unit 152 is included within Rasterizer 150. In a further alternate embodiment, Conflict Detection Unit 152 is included within Fragment Processor 155. Conflict Detection Unit 152 determines if a RAW conflict exists for each source read of a position in a buffer, as described further herein. Conflict Detection Unit 152 blocks processing of one or more fragments when the position conflict status indicates that a conflict exists. Conflict Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the fragment program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and buffer data (texture map, height field, bump map, shadow map, jitter values, and the like) from Local Memory 140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. Memory Controller 120 outputs the buffer data and the additional fragment program instructions to Read Interface 153. Texture Unit 154 outputs the buffer data, processed fragment data, and the additional fragment program instructions to a Fragment Processing Unit 156. Fragment Processing Unit 156 processes the processed buffer data and processed fragment data as specified by the additional fragment program instructions and outputs shaded fragment data, e.g., x, y, color, depth, configuration control, other parameters, to Raster Operation Unit 165. In some embodiments Fragment Processing Unit 156 is configured to process at least two fragments in parallel. Likewise, Conflict Detection Unit 152 and Read Interface 153 may also be configured to process at least two fragments in parallel. Raster Operation Unit 165 optionally processes the shaded fragment data according to the configuration control. A Write Interface 157 within Raster Operation Unit 165 writes the optionally processed shaded fragment data to a buffer stored in Local Memory 140 or Host Memory 112, via Memory Controller 120. Write Interface 157 also outputs write position information to Conflict Detection Unit 152 to update the status of position conflicts, as described further herein.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>6:4–55.</p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory.</p> <p>26:24–39.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>29 “The method of claim 21, wherein storing the first output data comprises: accumulating, in the memory, outputs of computational elements executed by the GPU in performing the first</p>	<p>Kirk discloses “[t]he method of claim 21, wherein storing the first output data comprises: accumulating, in the memory, outputs of computational elements executed by the GPU in performing the first computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 12[a][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>computation in the sequence of computations.”</p>	<p>supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>30[a] “The method of claim 21, further comprising: storing, in the memory, an output of a previous computation in the sequence of computations; and”</p>	<p>Kirk discloses “[t]he method of claim 21, further comprising: storing, in the memory, an output of a previous computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 12[a][ii], claim 21.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>30[b] “accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations.”</p>	<p>Kirk discloses “accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations.”</p> <p><i>See</i> 12[d][i], 12[d][ii].</p> <p><i>Kirk further discloses a GPU (e.g., Graphics Processor 105 and/or Programmable Graphics Processing Pipeline 1250) accessing the output of the previous computation during performance of another computation (e.g., by using the output of one computational cycle as the input for the next computational cycle):</i></p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data 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. 4:1–20.</p> <p>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. 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. 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. 8:22–30.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>31 “The method of claim 21, wherein performing the first computation comprises executing a plurality of computational elements representing a</p>	<p>Kirk discloses “[t]he method of claim 21, wherein performing the first computation comprises executing a plurality of computational elements representing a layer of neurons in an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[a][i].</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
layer of neurons in an artificial neural network.”	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
32 “The method of claim 31, wherein all neurons in the layer of neurons are described by the same equation.”	<p>Kirk discloses “The method of claim 31, wherein all neurons in the layer of neurons are described by the same equation.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[a][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
40[pre] “A system for executing an artificial neural network, the system comprising:”	<p>To the extent the preamble is limiting, Kirk discloses “[a] system for executing an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 21[pre].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[a] “a central processing unit (CPU) to provide first input data;”</p>	<p>Kirk discloses “a central processing unit (CPU) to provide first input data.” <i>See e.g.:</i></p> <p><i>See</i> 1[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[b] “a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network;”</p>	<p>Kirk discloses “a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][ii].</p> <p><i>As a non-limiting example, Kirk discloses a memory (e.g., Host Memory 112 and/or Local Memory 140) coupled to the CPU (e.g., CPU 102) that stores input data in a first partition. Kirk also discloses referencing partitions by pointers (e.g., identifying data by addresses for locations in graphics memory):</i></p> <p>FIG. 2B illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading including the steps illustrated in FIG. 2A. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. The fragment program specifies writing a shaded first fragment to the position within the buffer. In step 203 Conflict Detection Unit 152 receives the position. In one embodiment the position is represented as a pair of coordinates, e.g., (x,y), (s,t), (u,v), and the like, and the coordinates or portions of the coordinates are stored in Conflict Detection Unit 152. The coordinates may be represented relative to</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>a buffer or relative to a display. Coordinates represented within a buffer may be converted into coordinates within a display, e.g., screen coordinates, by applying coordinate offsets based on a position of the buffer within the display. In another embodiment the position is represented as an address for a location in graphics memory. In yet another embodiment the position includes a buffer identifier specifying which of several buffers the position is associated with. In still another embodiment, Conflict Detection Unit 152 identifies a region including the location and stores data, e.g. one or more bits, corresponding to the region. A region may represent several positions, where the positions may correspond to a region of an image, a region of an output buffer, a sequence of physical memory addresses in graphics memory, or the like. Conflict Detection Unit 152 may store data for several regions, depending on a predetermined resolution of the positions to be tracked.</p> <p>In step 205 Fragment Processor 155 begins shading the first fragment, as specified by the fragment program producing a shaded first fragment, several cycles or more later. The shaded first fragment is output to Raster Operation Unit 165. In step 207 Fragment Processing Pipeline 160 receives a second fragment associated with the position within the buffer. To produce a shaded second fragment, the fragment program specifies reading the shaded first fragment as source data.</p> <p>7:55–8:16.</p> <p>FIG. 3A is a conceptual diagram of a Buffer 320 displayed by a display device, e.g., monitor, projector, and the like. Data stored in Buffer 320 is displayed on Display 330. Additional buffers of arbitrary sizes may be displayed on Display 330. Each buffer may be positioned for display relative to Display 330. A Pixel 340 within Buffer 320 is associated with an x,y position relative to Display 330. In an alternate embodiment the x,y origin is in the lower left corner of Display 330.</p> <p>9:32–55.</p> <p>FIG. 3B illustrates a Portion of Graphics Memory 350 including locations storing data for Buffer 320. Locations within a Section 360 store data for Buffer 320. For example, a Location 366 stores data associated with Pixel 340, e.g., color, depth, stencil, shadow depth, and the like. An Address 364 is used to access the Location 366. Address 364 may be computed based on an x,y position and a base address corresponding to a first location within Section 360. In an alternate embodiment Address 364 is computed based on a position within Buffer 320 and an address offset within Portion of Graphics Memory 350 corresponding to Section 360. A Section 370 includes locations storing data for another</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

buffer. Each buffer is associated with a unique buffer identifier that may be used to determine a corresponding base address.
9:41–55.

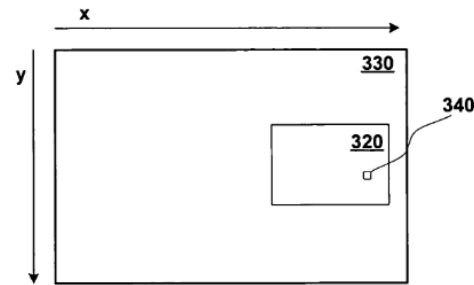


Fig. 3A

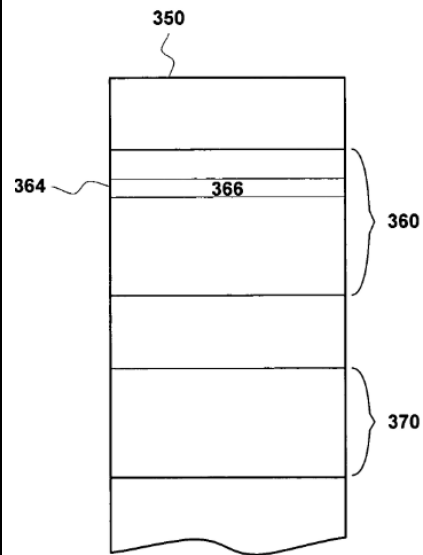


Fig. 3B

Figs. 3A, 3B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[c] “a processing unit, operably coupled to the memory, to perform, during computation of the first layer of neurons, at least one calculation on the first input data so as to generate first output data, the first output data representing an output of at least one neuron in the first layer of neurons; and”</p>	<p>Kirk discloses “a processing unit, operably coupled to the memory, to perform, during computation of the first layer of neurons, at least one calculation on the first input data so as to generate first output data, the first output data representing an output of at least one neuron in the first layer of neurons.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 21[d].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d] “a controller, operably coupled to the processing unit and the memory, to:”</p>	<p>Kirk discloses “a controller, operably coupled to the processing unit and the memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[d].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d][i] “store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network,”</p>	<p>Kirk discloses “store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 21[d], 40[b].</p> <p><i>As a non-limiting example, Kirk discloses a memory (e.g., Host Memory 112 and/or Local Memory 140) that stores output data in a second partition. Kirk also discloses referencing partitions by pointers (e.g., identifying data by addresses for locations in graphics memory). When using pointers, as disclosed in Kirk, this use of an output as a subsequent input is most efficiently performed by swapping the first pointer to output data with the second pointer to input data:</i></p> <p>Various embodiments of a method of the invention include processing fragment program instructions. A pixel load instruction including a source address corresponding to a location within the buffer is received. A write to the source address is determined to be pending. Data stored in the location corresponding to the source address is read after the write to the source address is complete. 1:48–54.</p> <p>FIG. 2B illustrates an embodiment of a method of detecting and avoiding RAW position conflicts during fragment shading including the steps illustrated in FIG. 2A. In step 201 Fragment Processing Pipeline 160 receives a first fragment associated with a position within a buffer. The fragment program specifies writing a shaded first fragment to the position within the buffer. In step 203 Conflict Detection Unit 152 receives the position. In one embodiment the position is represented as a pair of coordinates, e.g., (x,y), (s,t), (u,v), and the like, and the coordinates or portions of the coordinates are stored in Conflict Detection Unit 152. The coordinates may be represented relative to a buffer or relative to a display. Coordinates represented within a buffer may be converted into coordinates within a display, e.g., screen coordinates, by applying coordinate offsets based on a</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>position of the buffer within the display. In another embodiment the position is represented as an address for a location in graphics memory. In yet another embodiment the position includes a buffer identifier specifying which of several buffers the position is associated with. In still another embodiment, Conflict Detection Unit 152 identifies a region including the location and stores data, e.g. one or more bits, corresponding to the region. A region may represent several positions, where the positions may correspond to a region of an image, a region of an output buffer, a sequence of physical memory addresses in graphics memory, or the like. Conflict Detection Unit 152 may store data for several regions, depending on a predetermined resolution of the positions to be tracked. 7:55–8:16.</p> <p>FIG. 3B illustrates a Portion of Graphics Memory 350 including locations storing data for Buffer 320. Locations within a Section 360 store data for Buffer 320. For example, a Location 366 stores data associated with Pixel 340, e.g., color, depth, stencil, shadow depth, and the like. An Address 364 is used to access the Location 366. Address 364 may be computed based on an x,y position and a base address corresponding to a first location within Section 360. In an alternate embodiment Address 364 is computed based on a position within Buffer 320 and an address offset within Portion of Graphics Memory 350 corresponding to Section 360. A Section 370 includes locations storing data for another buffer. Each buffer is associated with a unique buffer identifier that may be used to determine a corresponding base address. 9:41–55.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
40[d][ii] “transfer the first output data to another memory during	Kirk discloses “transfer the first output data to another memory during computation of the second layer of neurons.” <i>See e.g.:</i>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>computation of the second layer of neurons, and”</p>	<p><i>See</i> 1[d][iv], 12[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d][iii] “dictate an order of execution of instructions to the processing unit to perform the computation of the first layer of neurons.”</p>	<p>Kirk discloses “dictate an order of execution of instructions to the processing unit to perform the computation of the first layer of neurons.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Kirk discloses a controller (e.g., Graphics Interface 117, Front End 1230, and/or Index Processor 1235) that dictates an order of instructions to a processing unit (e.g., Programmable Graphics Processing Pipeline 1250):</i></p> <p>FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 110 via Graphics Interface 117. Front End 1230 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235. Some of the formatted commands are used by a Programmable Graphics Processing Pipeline 1250 to initiate processing of data by providing the location of program instructions or graphics data stored in memory. Index Processor 1235, Programmable Graphics Processing Pipeline 1250 and Raster Operation Unit 165 each include an interface to Memory Controller 120 through which program instructions and data may be read from graphics memory. 26:24–39.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>41 “The system of claim 40, wherein the processing unit comprises a graphics processing unit.”</p>	<p>Kirk discloses “[t]he system of claim 40, wherein the processing unit comprises a graphics processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>42 “The system of claim 40, wherein the controller is configured to send instructions for performing the at least one calculation to the processing unit.”</p>	<p>Kirk discloses “[t]he system of claim 40, wherein the controller is configured to send instructions for performing the at least one calculation to the processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][ii].</p> <p><i>As a non-limiting example, Kirk discloses a controller (e.g., Graphics Interface 117, Front End 1230, and/or Index Processor 1235) that is configured to send instructions for performing at least one calculation to the processing unit (e.g., Programmable Graphics Processing Pipeline 1250):</i></p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within Fragment Processing Pipeline 160. Graphics Processing Pipeline 103 receives a stream of program instructions (vertex program instructions and fragment program instructions) and data from Graphics Interface 117 or Memory Controller 120, and performs vector floating-point operations or other processing operations using the data. The program instructions configure subunits within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160. The program instructions and data are stored in graphics memory. When a portion of Host Memory 112 is used to store program instructions and data, the portion of Host Memory 112 can be uncached so as to increase performance of access by Programmable Graphics Processor 105. Alternatively, configuration information is written to registers within Geometry Processor 130, Rasterizer 150 and Fragment Processing Pipeline 160 using program instructions, encoded with the data, or the like. 4:21–42.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>43[a] “The system of claim 40, wherein the memory further comprises: a third partition to store internal variables; and”</p>	<p>Kirk discloses “The system of claim 40, wherein the memory further comprises: a third partition to store internal variables.” <i>See e.g.:</i></p> <p><i>See</i> 6[a]; claims 28, 40.</p> <p><i>As a non-limiting example, Kirk discloses a memory (e.g., Host Memory 112 and/or Local Memory 140) that stores internal variables (e.g., graphics data) in a third partition (e.g., a logical partition in the memory):</i></p> <p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>3:47–67.</p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data 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.</p> <p>4:1–20.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>43[b] “a fourth partition to store data used as input at a particular layer of neurons of the artificial neural network.”</p>	<p>Kirk discloses “a fourth partition to store data used as input at a particular layer of neurons of the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 6[b], 43[a], claim 28.</p> <p><i>As a non-limiting example, Kirk discloses a memory (e.g., Host Memory 112 and/or Local Memory 140) that stores input data (e.g., graphics data) to be processed in a fourth partition (e.g., a logical partition in the memory):</i></p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>Host computer 110 communicates with Graphics Subsystem 107 via System Interface 115 and a Graphics Interface 117. Graphics Subsystem 107 includes a Local Memory 140 and a Programmable Graphics Processor 105. Programmable Graphics Processor 105 uses memory to store graphics data and program instructions, where graphics data is any data that is input to or output from computation units within Programmable Graphics Processor 105. Graphics memory is any memory used to store graphics data or program instructions to be executed by Programmable Graphics Processor 105. Graphics memory may include portions of Host Memory 112, Local Memory 140 directly coupled to Programmable Graphics Processor 105, register files coupled to the computation units within Programmable Graphics Processor 105, and the like. 3:32–46.</p> <p>In addition to Graphics Interface 117, Programmable Graphics Processor 105 includes a Graphics Processing Pipeline 103, a Memory Controller 120 and an Output Controller 180. Data and program instructions received at Graphics Interface 117 can be passed to a Geometry Processor 130 within Graphics Processing Pipeline 103 or written to Local Memory 140 through Memory Controller 120. Memory Controller 120 includes read interfaces and write interfaces that each generate address and control signals to Local Memory 140, storage resources, and Graphics Interface 117. Storage resources may include register files, caches, FIFO (first in first out) memories, and the like. In addition to communicating with Local Memory 140, and Graphics Interface 117, Memory Controller 120 also communicates with Graphics Processing Pipeline 103 and Output Controller 180 through read and write interfaces in Graphics Processing Pipeline 103 and a read interface in Output Controller 180. The read and write interfaces in Graphics Processing Pipeline 103 and the read interface in Output Controller 180 generate address and control signals to Memory Controller 120. 3:47–67.</p> <p>Within Graphics Processing Pipeline 105, Geometry Processor 130 and a programmable graphics fragment processing pipeline, Fragment Processing Pipeline 160, perform a variety of computational functions. Some of these functions are table lookup, scalar and vector addition, multiplication, division, coordinate-system mapping, calculation of vector normals, tessellation, calculation of derivatives, interpolation, and the like. Geometry Processor 130 and Fragment Processing Pipeline 160 are optionally configured such that data processing operations are performed in multiple passes</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>through Graphics Processing Pipeline 103 or in multiple passes through Fragment Processing Pipeline 160. Each pass through Programmable Graphics Processor 105, Graphics Processing Pipeline 103 or Fragment Processing Pipeline 160 concludes with optional processing by a Raster Operation Unit 165. Data 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. 4:1–20.</p> <p><i>Kirk further discloses Pixel Input Buffer and Vertex Input Buffer, which are partitions in Programmable Processing Pipeline 1250 that store data used as inputs:</i></p> <p>FIG. 12B is a block diagram of an exemplary embodiment of Programmable Graphics Processing Pipeline 1250 in accordance with one or more aspects of the present invention. Samples, such as surfaces, primitives, or the like, are received from Index Processor 1235 by Programmable Graphics Processing Pipeline 1250 and stored in a Vertex Input Buffer 1220 in a register file, FIFO (first in first out) memory, cache, or the like (not shown). The samples are broadcast to Execution Pipelines 1240, four of which are shown in FIG. 12B. An alternate embodiment may include either more or fewer Execution Pipelines 1240. Each Execution Pipeline 1240 includes at least one multithreaded processing unit. The samples output by Vertex Input Buffer 1220 may be processed by any one of the Execution Pipelines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available. 26:62–27:11.</p> <p>Pixel Input Buffer 1215 receives fragments from Raster Unit 1010 and outputs the fragments to each Execution Pipeline 1240. The fragments, output by Pixel Input Buffer 1215, are each processed (as in Fragment Processing Unit 156) by only one of the Execution Pipelines 1240. Pixel Input Buffer 1215 determines which one of the Execution Pipelines 1240 to output each fragment to depending on a position, e.g., (x,y), associated with each sample. In this manner, each fragment is output to the Execution Pipeline 1240 designated to process fragments associated with the position. 28:10–20.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438

Kirk

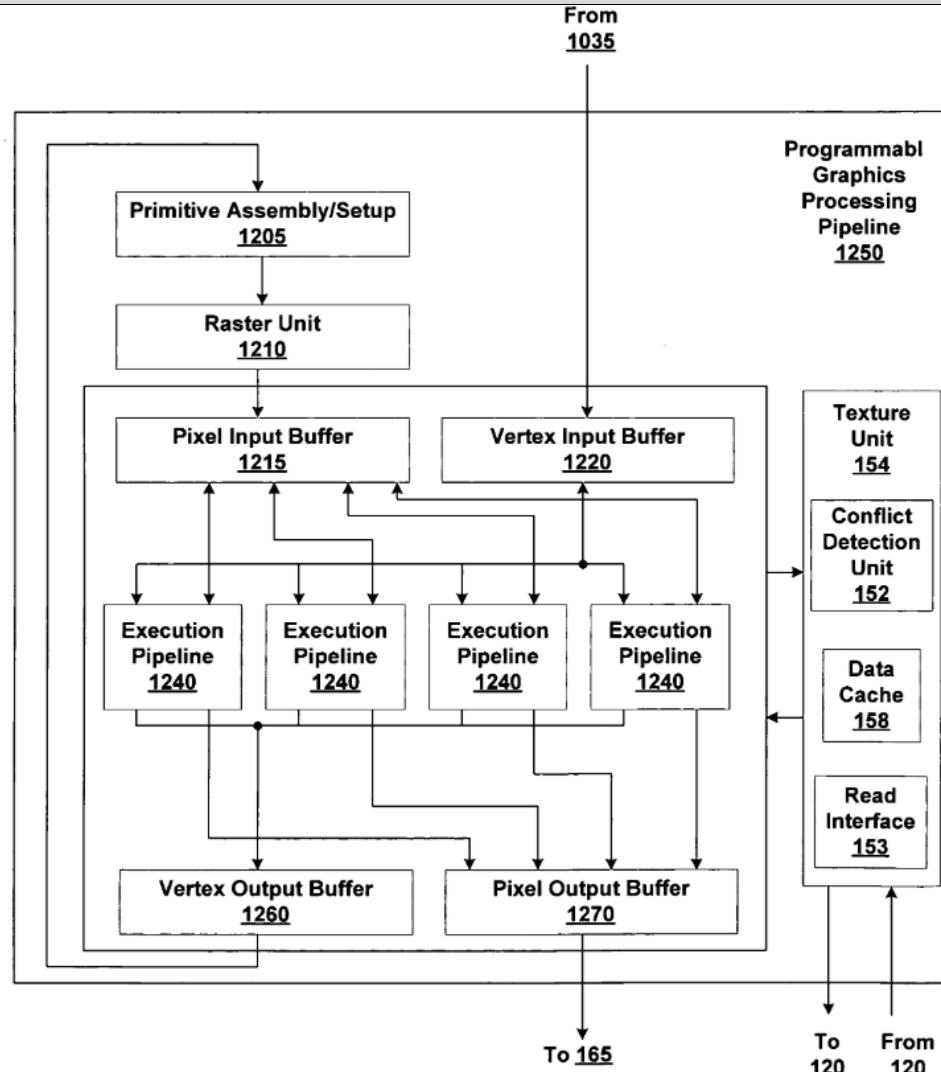


Fig. 12B

FIG. 12B.

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[pre] “A computer system, comprising:”</p>	<p><i>See</i> 1[pre].</p>
<p>44[a] “a central processing unit to receive input data acquired from an external system;”</p>	<p>Kirk discloses “a central processing unit to receive input data acquired from an external system.” <i>See e.g.:</i></p> <p><i>See</i> 1[a], 21[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[b] “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit;”</p>	<p>Kirk discloses “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[c] “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory, the accelerator comprising:”</p>	<p>Kirk “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[c][i] “at least one processing unit to perform a sequence of computations representing an artificial neural network on the at least a portion of the input data so as to generate output data, intermediate computations in the sequence of computations representing layers of the neural network and yielding intermediate results; and”</p>	<p>Kirk discloses “at least one processing unit to perform a sequence of computations representing an artificial neural network on the at least a portion of the input data so as to generate output data, intermediate computations in the sequence of computations representing layers of the neural network and yielding intermediate results.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>44[c][ii] “accelerator memory, operably coupled to the at least one processing unit, to store the results of the sequence of computations; and”</p>	<p>Kirk discloses “accelerator memory, operably coupled to the at least one processing unit, to store the results of the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d] “a controller, operably coupled to the at least one processing unit and the accelerator memory,”</p>	<p>Kirk discloses “a controller, operably coupled to the at least one processing unit and the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[d].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d][i] “to control transfer of the at least a portion of the input data into the accelerator memory during performance of the intermediate computations</p>	<p>Kirk discloses “to control transfer of the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii].</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>in the sequence of computations by the at least one processing unit,”</p>	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d][ii] “to control transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit, and”</p>	<p>Kirk discloses “to control transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d][iii] “to control performance of the sequence of computations by the at least one processing unit.”</p>	<p>Kirk discloses “to control performance of the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>45 “The computer system of claim 44, wherein the central processing unit is configured to receive the input data in response to a user interaction.”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein the central processing unit is configured to receive the input data in response to a user interaction.” <i>See e.g.:</i></p> <p><i>See</i> claim 2.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>46[a] “The computer system of claim 44, wherein: the central processing unit is configured to receive the input data at a first rate; and”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein: the central processing unit is configured to receive the input data at a first rate.” <i>See e.g.:</i></p> <p><i>See</i> 3[a], claim 44.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>46[b] “the at least one processing unit is configured to perform the</p>	<p>Kirk discloses “the at least one processing unit is configured to perform the sequence of computations at a second rate different than the first rate.” <i>See e.g.:</i></p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>sequence of computations at a second rate different than the first rate.”</p>	<p><i>See</i> 3[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>47 “The computer system of claim 44, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> claim 4.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>48 “The computer system of claim 44, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> claim 5.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
	<p>supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>49[a] “The computer system of claim 44, wherein accelerator memory comprises: a first memory partition to store parameters common to all of the computations in the sequence of computations; and”</p>	<p>Kirk discloses “The computer system of claim 44, wherein accelerator memory comprises: a first memory partition to store parameters common to all of the computations in the sequence of computations.” <i>See e.g.</i>:</p> <p><i>See</i> 6[a]; claims 26, 44.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>49[b] “a second memory partition to store data specific to at least one computation in the sequence of computations.”</p>	<p>Kirk discloses “a second memory partition to store data specific to at least one computation in the sequence of computations.” <i>See e.g.</i>:</p> <p><i>See</i> 6[b]; claim 27.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>50 “The computer system of claim 44, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.” <i>See e.g.:</i></p> <p><i>See claim 7.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>51 “The computer system of claim 44, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one processing unit has begun to perform another sequence of computations.”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one processing unit has begun to perform another sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 8.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>52 “The computer system of claim 51, wherein the controller is configured to</p>	<p>Kirk discloses “[t]he computer system of claim 51, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
<p>initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one processing unit.”</p>	<p>in parallel with performance of at least one computation in the other sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>53 “The computer system of claim 44, wherein the controller is configured to control execution of the sequence of computations by the at least one processing unit.”</p>	<p>Kirk discloses “[t]he computer system of claim 44, wherein the controller is configured to control execution of the sequence of computations by the at least one processing unit.”</p> <p><i>See</i> claim 10.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>54 “The computer system of claim 44, further comprising: at least one of a video camera, a microphone, or a cell recording electrode,</p>	<p>Kirk discloses “[t]he computer system of claim 44, further comprising: at least one of a video camera, a microphone, or a cell recording electrode, operably coupled to the central processing unit, to acquire the input data in real time.” <i>See e.g.:</i></p> <p><i>See</i> 1[a].</p>

Ex. B9 – Invalidity of U.S. Patent No. RE48,438 – Kirk

U.S. Patent No. RE48,438	Kirk
operably coupled to the central processing unit, to acquire the input data in real time.”	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Kirk and the knowledge of a POSITA, it would have been obvious to combine the teachings of Kirk with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Kirk. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.