Ex. A9 – Invalidity of U.S. Patent No. 8,648,867 – 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. 8,648,867 (the "'867 patent") under at least pre-AIA 35 U.S.C. § 102(e). Kirk anticipates and/or renders obvious claims 16–19 of the '867 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.

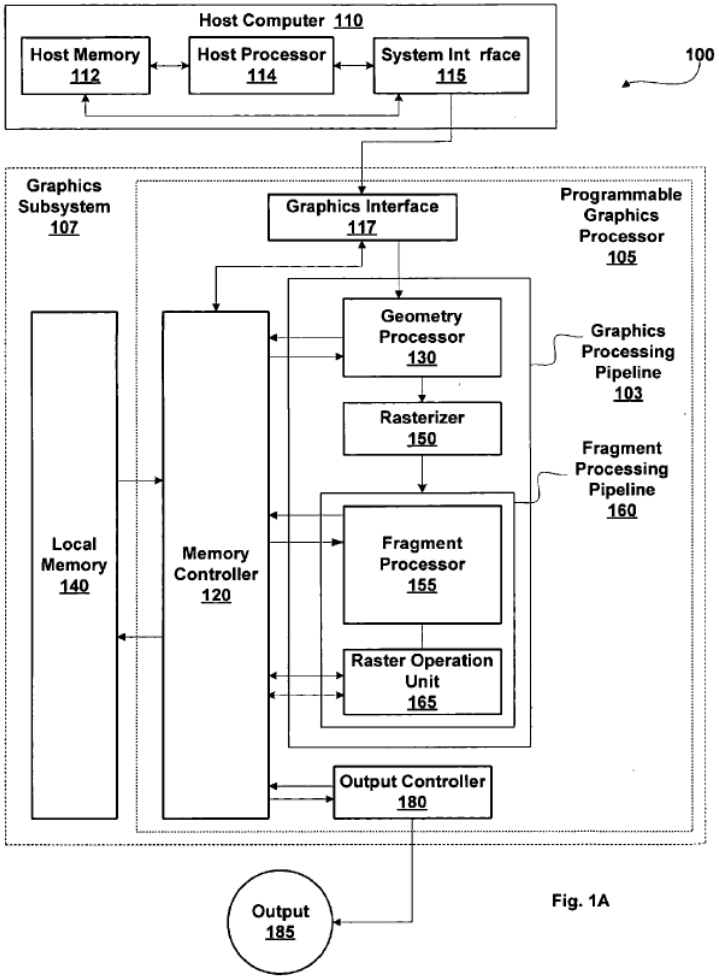
It is NVIDIA's position that Plaintiff's Infringement Contentions have not established that any accused product or service infringes any valid claim. Thus, NVIDIA's statements below should not be treated as an admission, implication, or suggestion that NVIDIA agrees with Plaintiff regarding either the 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.

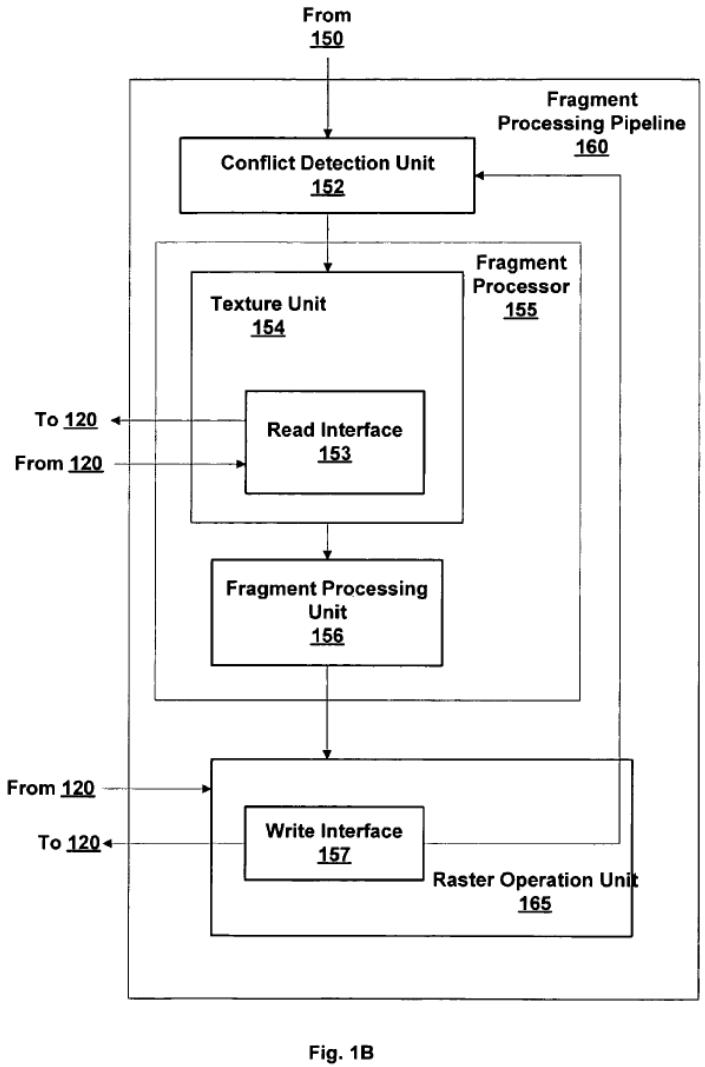
To the extent Plaintiff argues that any element below is not disclosed by Kirk, a person of ordinary skill in the art would have found it obvious to combine the teachings of Kirk with the background knowledge of a person of ordinary skill in the art and/or the additional references, and exemplary teachings, 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.

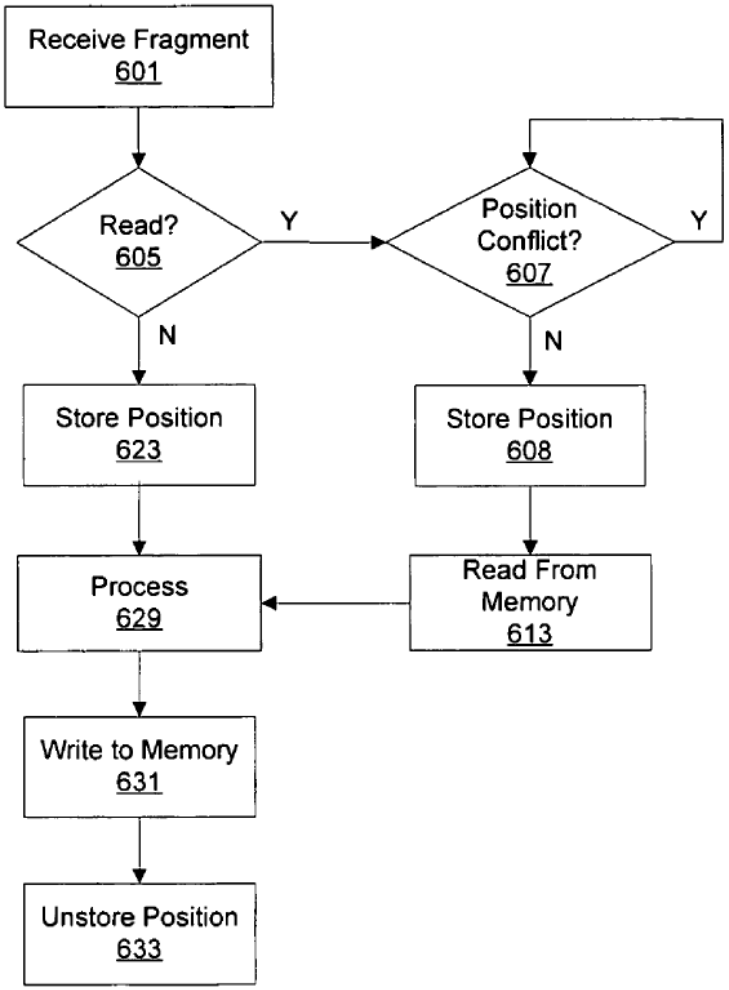
Plaintiff has yet to identify any limitation of the Asserted Claims that it contends is not anticipated and/or rendered obvious by Kirk. NVIDIA therefore expressly reserves the right to respond to any such contention, including by identifying additional obviousness combinations, if Plaintiff makes any 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.

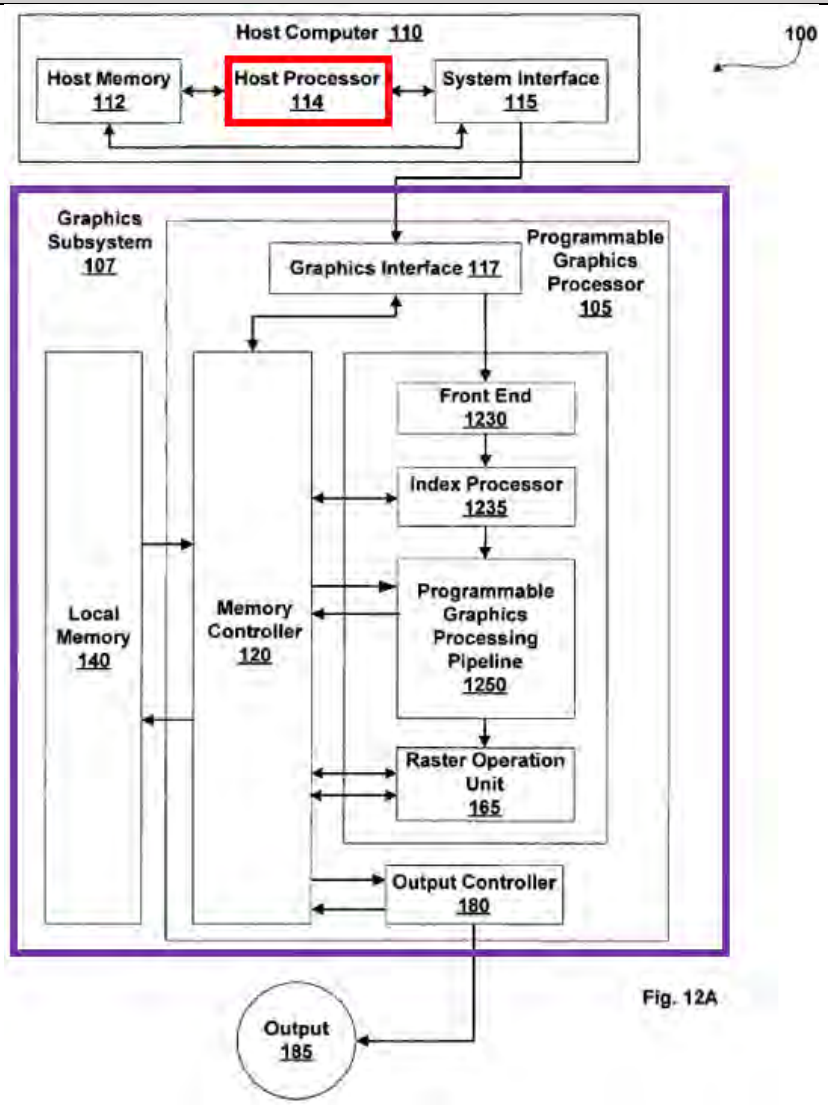
Neural AI, LLC, Exhibit 2013
Page 2013 - 1
IPR2025-00606, NVIDIA Corporation v. Neural AI, LLC

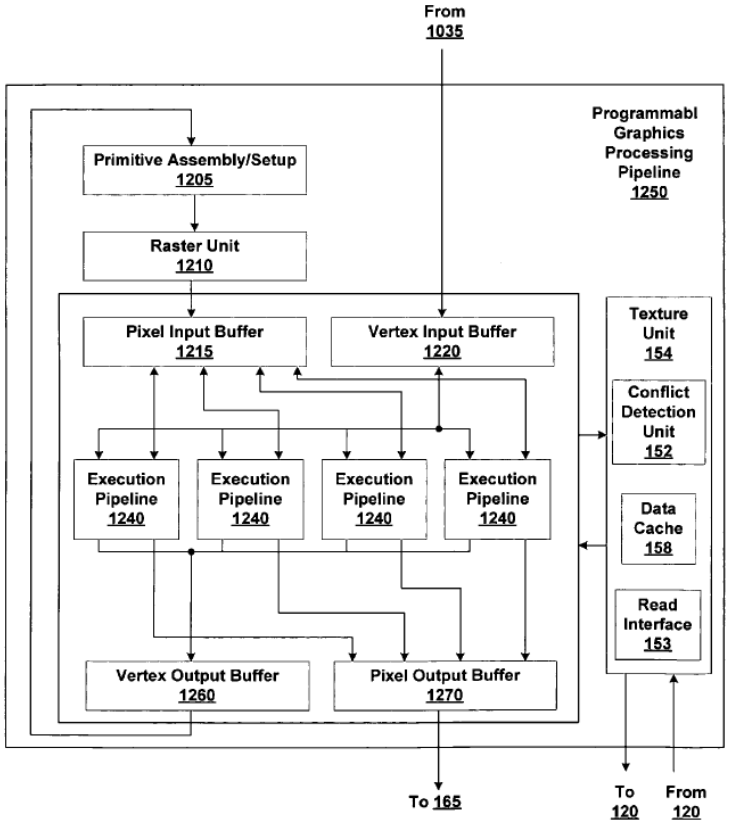
U.S. Patent No. 8,648,867	Kirk
16[pre]: A method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator, the method comprising:	To the extent the preamble is limiting, Kirk discloses "[a] method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator." *See e.g.*: *As a non-limiting example, Kirk discloses a method for performing a numerical simulation on input data (e.g., processing of data) in a computer system (e.g., a computing system) including a central processing unit (e.g., Host Processor 114) and an accelerator (e.g., Graphics Subsystem 107).*

U.S. Patent No. 8,648,867	Kirk
	 Fig. 1A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 1B Fig. 1B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 6A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated).

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12B Fig. 12B. 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

U.S. Patent No. 8,648,867	Kirk
	entered in conflict detection unit. During processing, the conflict detection unit is updated when a pending write to a destination location is completed. Abstract. 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. FIG. lA 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. 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.

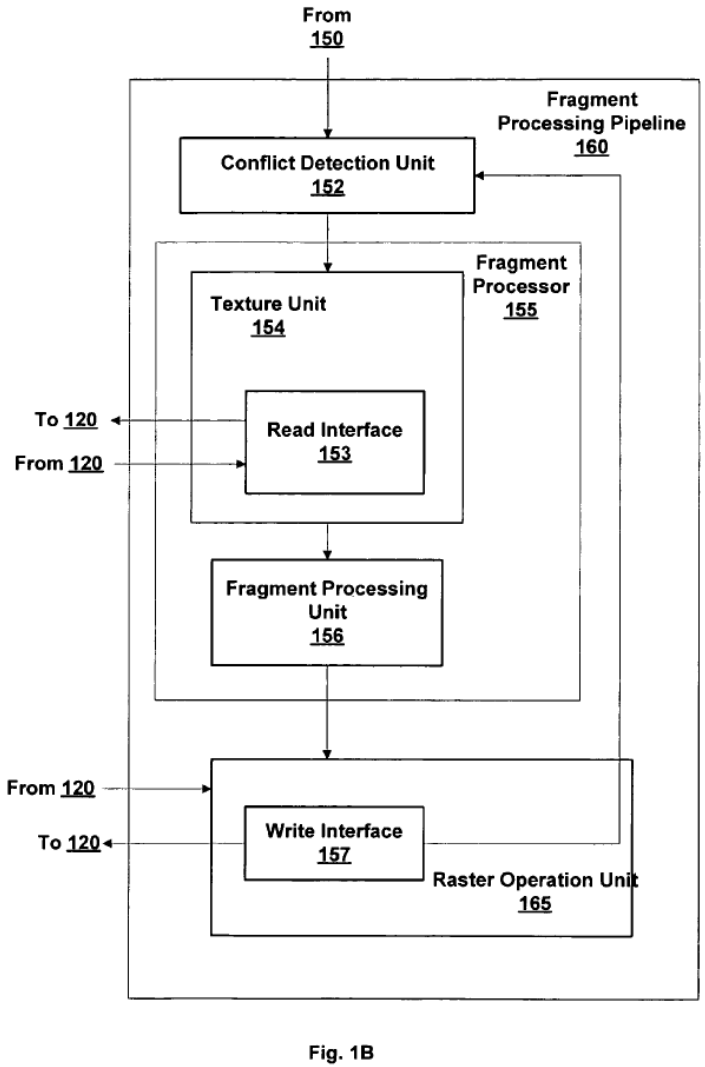
U.S. Patent No. 8,648,867	Kirk
	Vertex programs are sequences of vertex program instructions compiled by Host Processor 114 for execution within Geometry Processor 130 and Rasterizer 150. Fragment programs are sequences of fragment program instructions compiled by Host Processor 114 for execution within 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. 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. 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

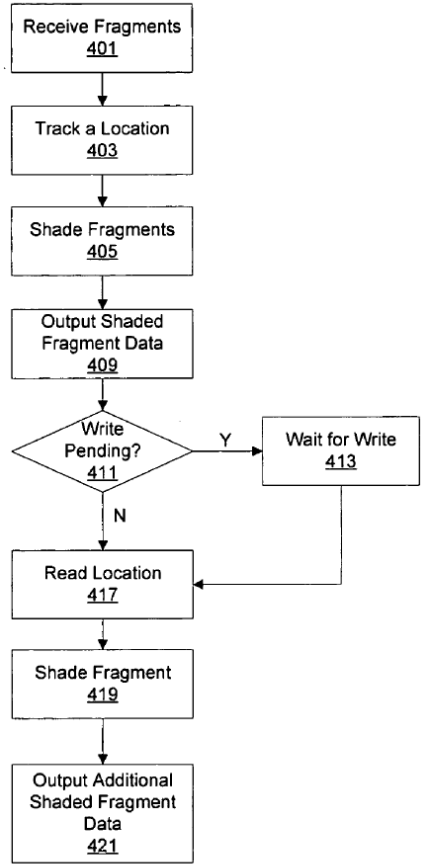
U.S. Patent No. 8,648,867	Kirk
	Detection Unit 152 outputs the fragment program instructions to Fragment Processor 155. Conflict Detection Unit 152 outputs fragment data for which conflicts do not exist to Fragment Processor 155. The fragment data is processed by Fragment Processor 155 according to the 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. 5:60–6:33. 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. 7:7–29. 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, then in step 211 Conflict Detection Unit 152 locks processing of the second fragment. In step 214 Raster Operation Unit 165 writes the shaded first fragment to the position in the buffer stored in graphics memory. Raster Operation Unit 165 outputs the write position information to Conflict Detection Unit 152 confirming that the

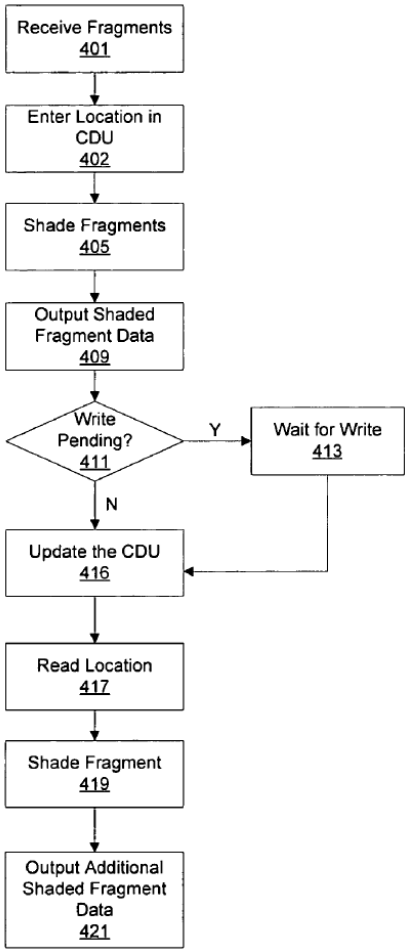
U.S. Patent No. 8,648,867	Kirk
	write is complete. In step 215 the position is removed from Conflict Detection Unit 152, thereby updating the status of the position conflict. 8:26–36. FIG. 6A illustrates a method of processing fragments in Fragment Processing Pipeline 160. In step 601 Conflict Detection Unit 152 receives a fragment. In step 605, Conflict Detection Unit 152 determines if processing the fragment as specified by a fragment program requires a read of source data, and, if not, Conflict Detection Unit 152 proceeds to step 621. If, in step 605, Conflict Detection Unit 152 determines that the fragment program specifies a read, in step 607 Conflict Detection Unit 152 determines if there is a position conflict with the location to be read (source location). If, in step 607 Conflict Detection Unit 152 determines there is a position conflict, Conflict Detection Unit 152 remains in step 607 and processing of the fragment does not continue until the conflict does not exist. The conflict does not exist when a pending write to the location is completed. 11:52–67. When the destination write confirmation is received from Write Interface 157, Input 701 provides the destination write 45 confirmation to Hash Function 715. Hash Function 715 outputs an index to Storage 720 and the entry corresponding to the index is cleared or otherwise marked as available. When the source read request is asserted, Storage 720 outputs the data stored in the entry corresponding to the index to a Compare 725. Compare 725 also receives the second portion of Input 701 from Hash Function 715 and compares it to the data output by Storage 720. When the second portion of Input 701 is equal to the data output by Storage 720, Compare 725 outputs a signal on Output 703 indicating that a position conflict exists. When the second portion of Input 701 is not equal to the data output by Storage 720 Compare 725 outputs a signal indicating that a position conflict does not exist. When a position conflict exists, Conflict Detection Unit 152 blocks processing of the fragment until the position conflict does not exist, i.e., a pending write to the source read position is completed. 13:44–62.

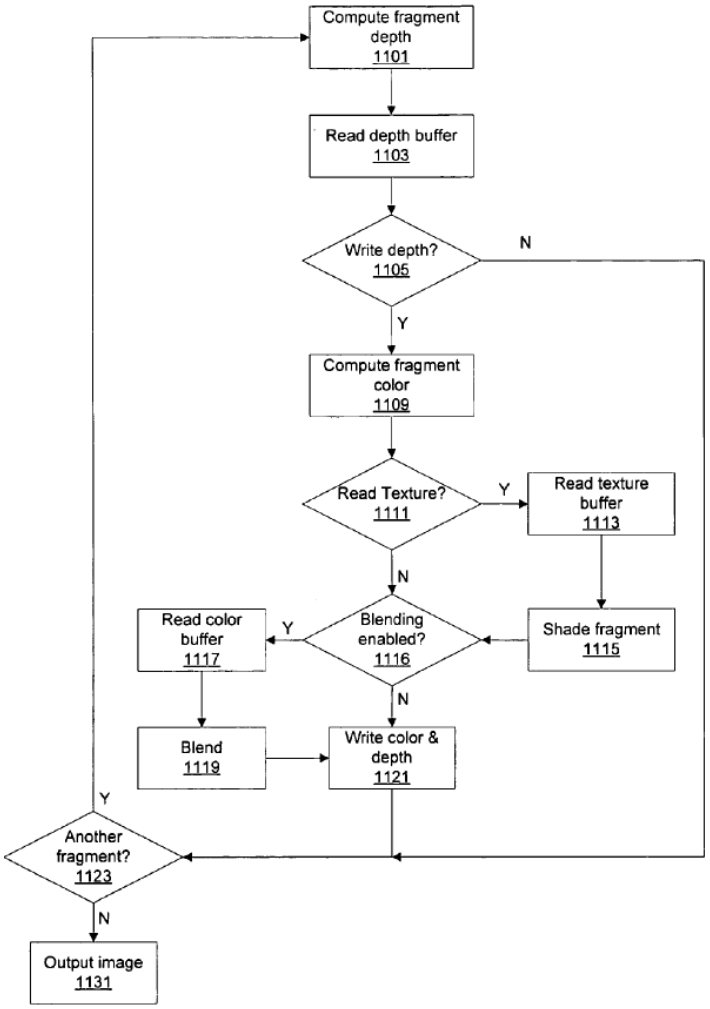
U.S. Patent No. 8,648,867	Kirk
	In an embodiment the source location specified in the PLD instruction is the first destination location specified in the fragment program instruction received in step 1015. Execution of the PLD instruction in the embodiment permits reading the source location during processing 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. 19:48–57. In step 1123, Fragment Processor 155 determines if another fragment will be processed, and, if so, returns to step 1101. In step 1101 Fragment Processor 155 is configured by fragment program instructions to compute the fragment depth value of another fragment for another position. In step 1103 Texture Unit 154 reads a depth buffer value for the other position. The read is performed using the PLD instruction, and the depth buffer value read is stored in Register 159. Because Fragment Processor 155 is pipelined, processing of the other fragment may begin before processing of the fragment is completed. Therefore, the PLD instruction should be used to detect and avoid RAW position conflicts. When the PLD instruction is not used, Fragment Processing Pipeline 160 may be flushed using an instruction to assure processing of the other fragment does not begin until the processed fragment is output to graphics memory. Flushing should be minimized to effectively utilize computing resources within Fragment Processing Pipeline 160 and to maximize fragment throughput. 21:37–55. 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.

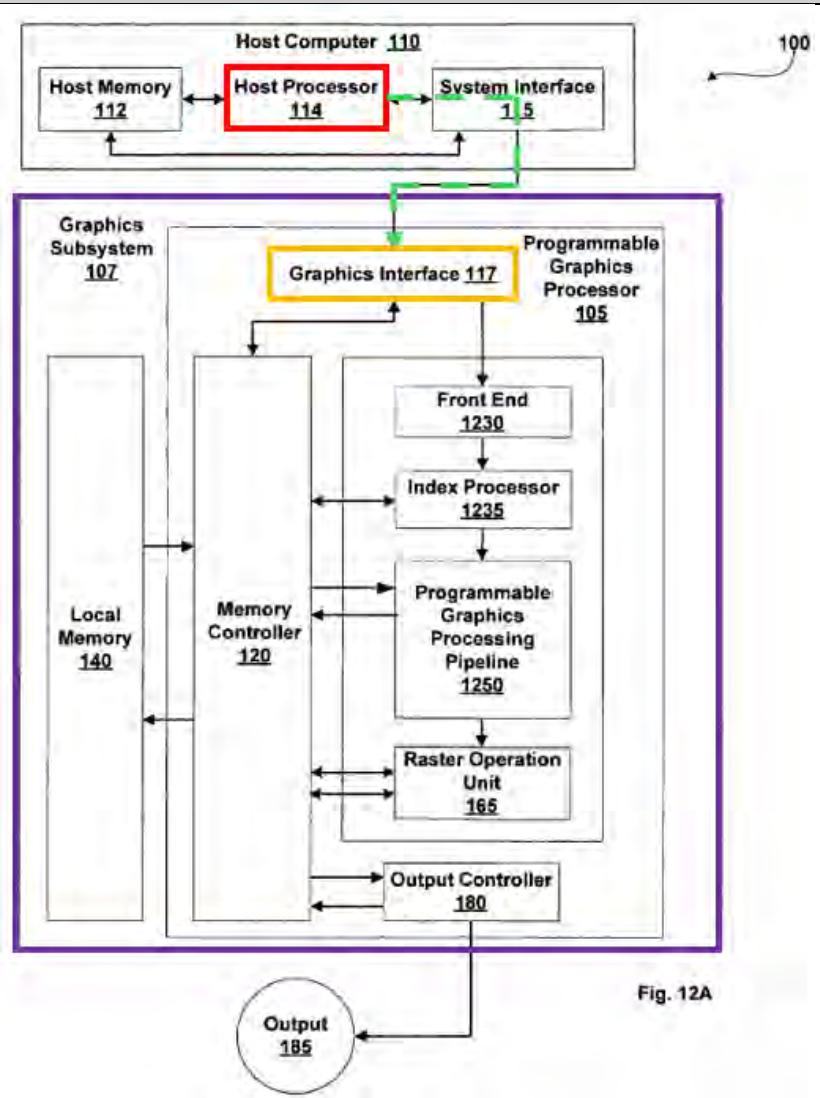
U.S. Patent No. 8,648,867	Kirk
	26:24–39. In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 26:56–61. To the extent Plaintiff contends that the preamble 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 '867 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 A1–A13, and Appendix A.
16[a]: receiving, by an accelerator, first input data from the central processing unit;	Kirk discloses "receiving, by an accelerator, first input data from the central processing unit." *See e.g.*: *As a non-limiting example, Kirk discloses an accelerator (e.g., Graphics Subsystem 107 at Graphics Interface 117), that receives first input data (e.g., graphics data) from the central processing unit (e.g., Host Processor 114).*

U.S. Patent No. 8,648,867	Kirk
	 Fig. 1B Fig. 1B.

U.S. Patent No. 8,648,867	Kirk
	 **Fig. 4A** Fig. 4A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 4B Fig. 4B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 11A Fig. 11A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated).

U.S. Patent No. 8,648,867	Kirk
	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. 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. 3:32–67. 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

U.S. Patent No. 8,648,867	Kirk
	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. 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. 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

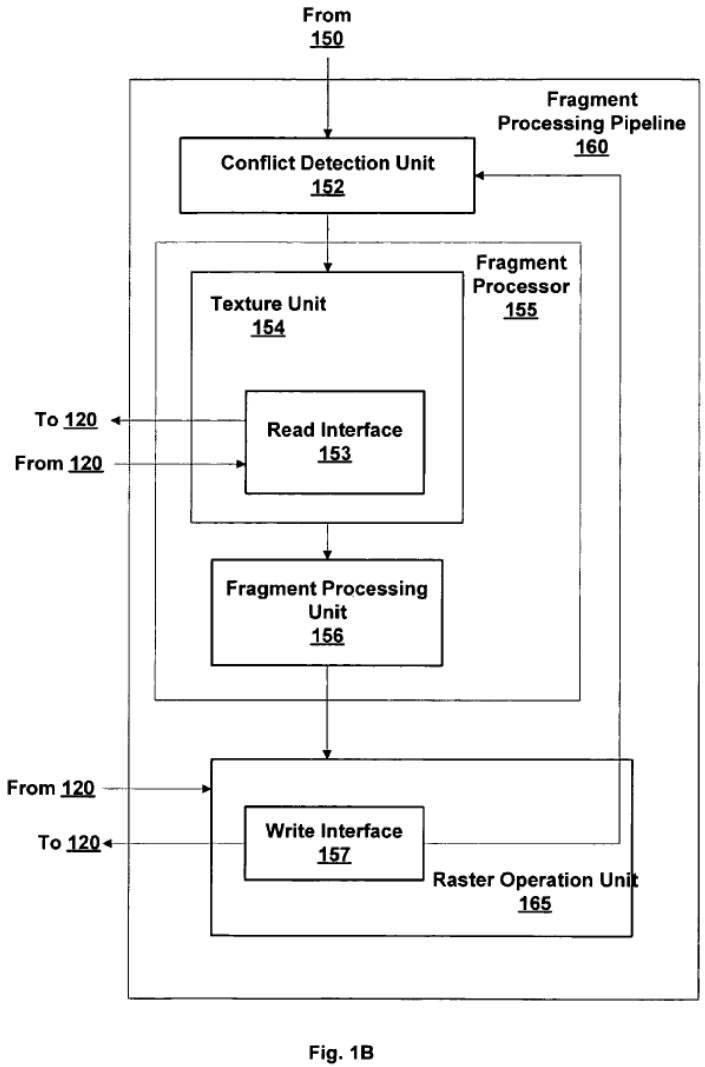
U.S. Patent No. 8,648,867	Kirk
	140 or Host Memory 112, via Memory Controller 120. The buffer data stored in graphics memory may be generated by Programmable Graphics Processor 105, by Host Processor 114, by another device, by a human, or the like. 5:60–6:33. In step 209 Conflict Detection Unit 152 determines 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. FIG. 4A illustrates a method of processing graphics data (fragments) in accordance with one or more aspects of the present invention. Fragments are not necessarily processed in the order in which the fragments are received, specifically when a position conflict exists since one or more fragments for which conflicts do not exist may bypass one or more fragments for which conflicts do exist. However, fragment order for each x,y position is maintained, i.e. a fragment associated with one x,y position may bypass a fragment associated with another x,y position. In step 401 fragments are received by Fragment Processing Pipeline 160. A fragment is associated with a specific location in a buffer to be read. Other fragments may be associated with either, the specific location in the buffer, other locations in the buffer, or locations in another buffer. In step 403 the location in the buffer associated with the fragment is tracked using Conflict Detection Unit 152, as described further herein. Conflict Detection Unit 152 determines that a write to the location in the buffer is pending. In step 405 at least a portion of the fragments are shaded by Fragment

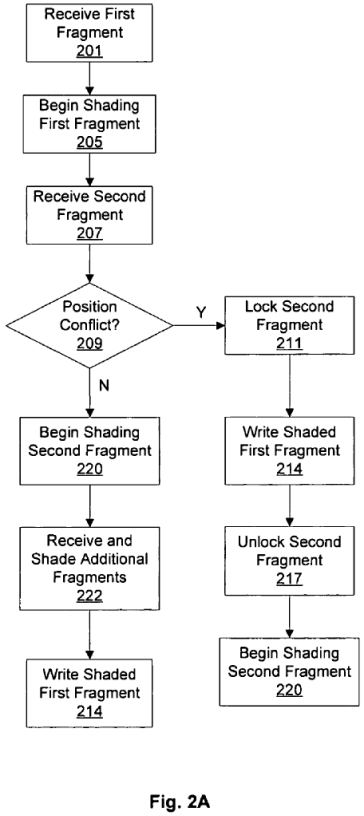
U.S. Patent No. 8,648,867	Kirk
	Processing Unit 156 as specified by a fragment program, producing shaded fragment data. In step 409 the shaded fragment data is output by Fragment Processor 155 to Raster Operation Unit 165. 9:56–10:11. FIG. 4B illustrates a method of processing graphics data including 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. 10:39–45. FIG. 11A illustrates an embodiment of a method of depth buffering prior to shading graphics data in accordance with one or more aspects of the present invention. It is desirable to minimize texture reads and computing resources by not shading any fragments that fail the depth test, i.e., which are behind another fragment. Therefore, to maximize performance and minimize unnecessary graphics memory reads it is desirable to perform depth buffering prior to shading. The PLD instruction may be used to configure Fragment Processor 155 to perform depth buffering prior to shading. 20:40–49. In step 1105, Fragment Processing Unit 156 receives the depth buffer value read in step 1103 and the fragment depth value computed in step 1101. Fragment Processing Unit 156 is configured by the fragment program instructions to determine if the fragment depth value will be written to the depth buffer. If, in step 1105, Fragment Processing Unit 156 determines the fragment depth value will not be written to the depth buffer, then, in step 1123 Fragment Processor 155 determines if another fragment will be processed, and, if not, in step 1131 Output Controller 180 reads the color buffer for output to Output 185, such as a display or the like. In an alternative embodiment, the color buffer is output via either Graphics Interface 117 or Output Controller 180 to a film recording device or written to a peripheral device, e.g., disk drive, tape, compact disk, or the like. 21:56–22:3. If, in step 1125 Graphics Processing Pipeline 103 determines another pass through Graphics Processing Pipeline 103 will not be completed, then in step 1131, Output Controller 180 reads the

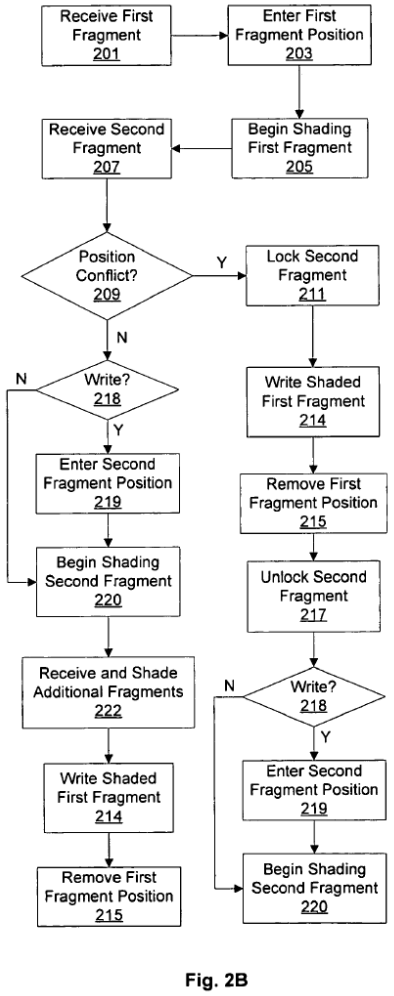
U.S. Patent No. 8,648,867	Kirk
	first or second color buffer, whichever was written during the last pass, for output to Output 185, such as a display or the like. In an alternative embodiment, the color buffer is output via either Graphics Interface 117 or Output Controller 180 to a film recording device or written to a peripheral device, e.g., disk drive, tape, compact disk, or the like. 24:32–41. In step 1128 Raster Operation Unit 165 writes the blended color values to an output buffer (color buffer). The output buffer may be one of the color buffers read in step 1124. In step 1132 Output Controller 180 reads the output buffer for output to Output 185, such as a display or the like. In an alternative embodiment, the output buffer is output via either Graphics Interface 117 or Output Controller 180 to a film recording device or written to a peripheral device, e.g., disk drive, tape, compact disk, or the like. In further embodiments of methods of processing graphics data, the PLD instruction may be used to perform frame buffer blending, floating-point raster operations, dual depth shadow mapping, and the like, using Fragment Processing Pipeline 160. 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. 26:10–39. 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

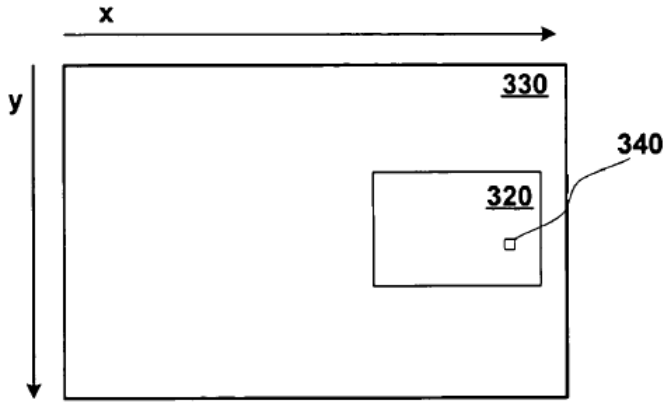
U.S. Patent No. 8,648,867	Kirk
	representation (object space) to an alternatively based coordinate system such as world space or normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics 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. 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 35 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. 27:12–39. 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 '867 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 A1–A13, and Appendix A.
16[b]: transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator	Kirk discloses "transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation." *See e.g.*:

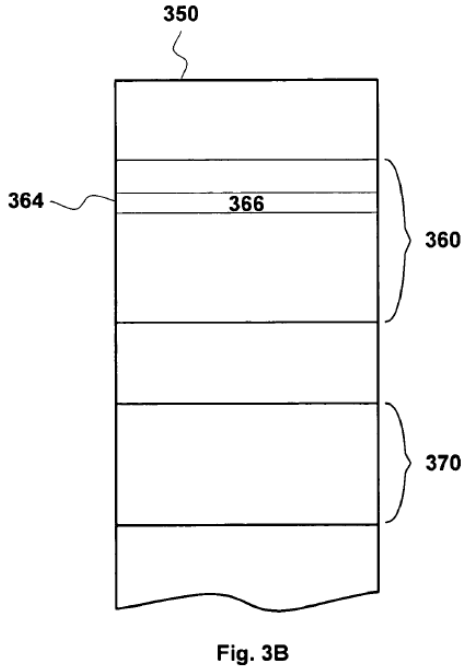
U.S. Patent No. 8,648,867	Kirk
memory before a first computational cycle of the numerical simulation;	*As a non-limiting example, Kirk discloses an accelerator controller (e.g., Graphics Interface 117, Front End 1230, and Index Processor 1235, and/or Memory Controller 120) and an accelerator memory (e.g., Local Memory 140). Kirk further discloses transferring the first input data into a "first partition" (e.g., a region including the location and stored data in Vertex Output Buffer 1260 and/or Texture Unit 154) before a first computational cycle of the numerical simulation (e.g., a fragment program). Kirk also discloses referencing partitions by pointers (e.g., accessing data using an address, which can be stored in a pointer, for a first location in graphics memory).*

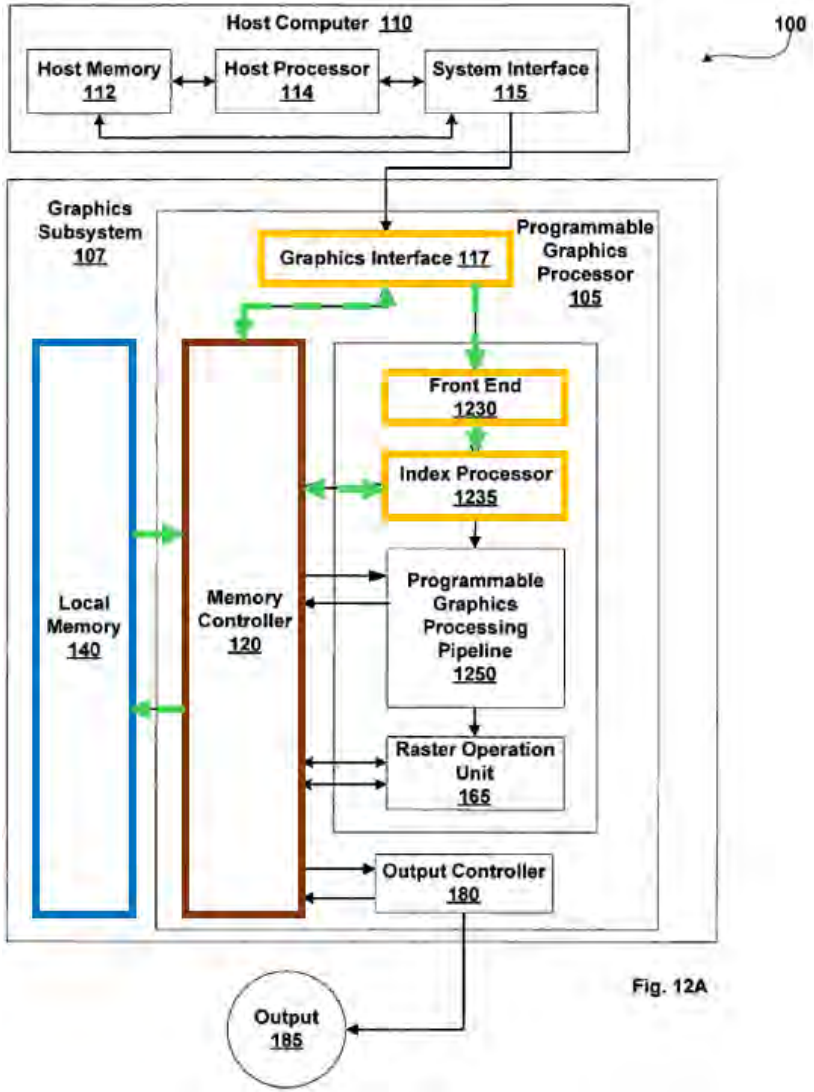
U.S. Patent No. 8,648,867	Kirk
	 Fig. 1B Fig. 1B.

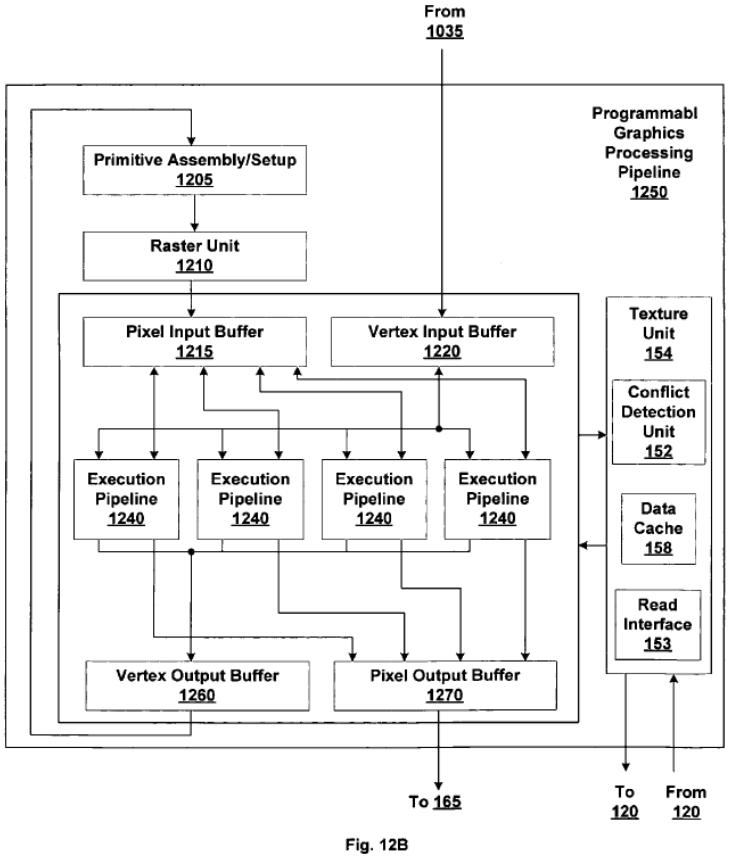
U.S. Patent No. 8,648,867	Kirk
	 Fig. 2A Fig. 2A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 2B Fig. 2B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 3A Fig. 3A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 3B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated).

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12B Fig. 12B. 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.

U.S. Patent No. 8,648,867	Kirk
	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. 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. 3:32–67. 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

U.S. Patent No. 8,648,867	Kirk
	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. 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 5 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 10 associated with the output data. 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

U.S. Patent No. 8,648,867	Kirk
	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. 4:57–5:32. Conventionally, a shading pipeline, such as Fragment Processing Pipeline 160, is flushed (using a flush instruction) prior to executing fragment program instructions that read the shadow map to avoid any read-after-write (RAW) conflicts. A RAW conflict exists when a write to an (x,y) position within a buffer is pending when a read from the same (x,y) position in the buffer is received. The RAW conflict is removed when the write to the (x,y) position within the buffer is completed. In various embodiments of Fragment Processing Pipeline 160 in accordance with one or more aspects of the present invention Fragment Processing Pipeline 160, a flush is not needed between writing a buffer and reading the buffer as an input to a Fragment Processor 155, as described further herein. 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. 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

U.S. Patent No. 8,648,867	Kirk
	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. 5:46–6:55. 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

U.S. Patent No. 8,648,867	Kirk
	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 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. 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 218 Conflict Detection Unit 152 determines if the fragment program specifies writing the shaded second fragment to the position within the buffer. If the fragment program specifies writing the shaded second fragment to the position within the buffer then, in step 219 the position is entered in Conflict Detection Unit 152 and the second fragment is output to Fragment Processor 155. In step 220 Fragment Processor 155 begins shading the second fragment as specified by the fragment program, producing a shaded second fragment several or more clock cycles later. Steps 212 and 213 proceed as previously described. Steps 222 and 214 proceed as previously described in relation to FIG. 2A and in step 215 the first position is removed from Conflict Detection Unit 152. 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.

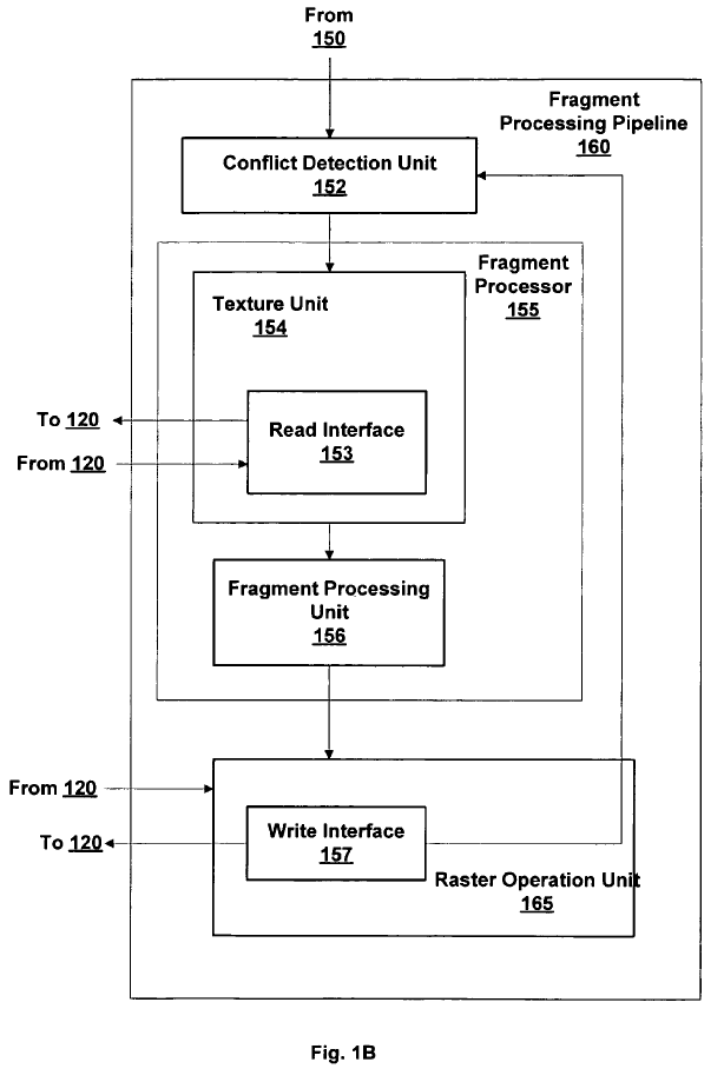
U.S. Patent No. 8,648,867	Kirk
	FIG. 3B illustrates a Portion of Graphics Memory 350 including locations storing data for Buffer 320. Locations within a Section 360 store data for Buffer 320. For example, a Location 366 stores data associated with Pixel 340, e.g., color, depth, stencil, shadow depth, 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. 9:16–55. If, in step 607, Conflict Detection Unit 152 determines there is not a position conflict, then in step 608 Conflict Detection Unit 152 stores a destination location and outputs the fragment to Fragment Processor 155. A destination location may be an x,y position, an address corresponding to a memory location, a register address, or the like. The destination position is stored because there is an implied write for each fragment being processed. However a processed fragment may not be written to a destination position if the processed fragment fails a depth or stencil test as determined by Raster Operation Unit 165. In step 613 Read Interface 153 reads the source location in graphics memory to obtain buffer data and outputs the buffer data. If, in step 605, Conflict Detection Unit 152 determines that the fragment program does not specify a read, in step 623 Conflict Detection Unit 152 stores the position to be written (destination location) in Conflict Detection Unit 152 and outputs the fragment to Fragment Processor 155. 12:1–18. A storage resource, Tile Lookup Unit 850 receives the tile origin included in the x,y position and outputs a corresponding tile number to a Control Unit 860. In an alternate embodiment the tile number equals the tile origin, Tile Lookup Unit 850 is omitted, and the tile origin is received by Control Unit 860. When Tile Lookup Unit 850 does not include an entry for each tile within an image or Display 330, i.e. the tile resolution exceeds the number of entries in Tile Lookup Unit 850, Tile Lookup Unit 850 outputs a match signal indicating whether or not the tile origin matches the contents of an entry. When the match signal indicates the tile origin matches the contents of an

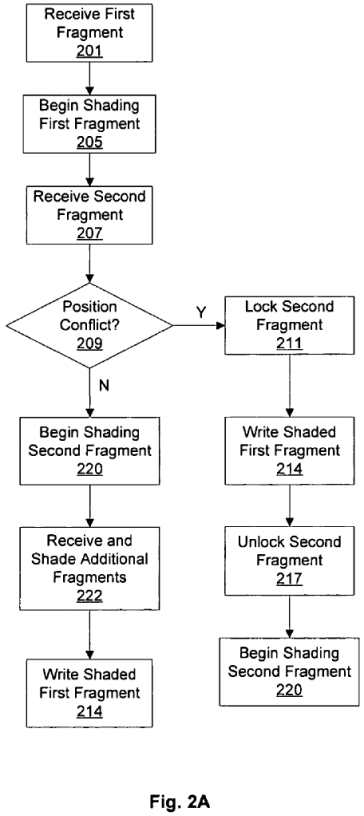
U.S. Patent No. 8,648,867	Kirk
	entry, Tile Lookup Unit 850 also outputs a tile number corresponding to the entry, e.g., entry address, to Control Unit 860. In one embodiment Tile Lookup Unit 850 is a content addressable memory (CAM). 14:52–67. In step 1105, Fragment Processing Unit 156 receives the depth buffer value read in step 1103 and the fragment depth value computed in step 1101. Fragment Processing Unit 156 is configured by the fragment program instructions to determine if the fragment depth value will be written to the depth buffer. If, in step 1105, Fragment Processing Unit 156 determines the fragment depth value will not be written to the depth buffer, then, in step 1123 Fragment Processor 155 determines if another fragment will be processed, and, if not, in step 1131 Output Controller 180 reads the color buffer for output to Output 185, such as a display or the like. In an alternative embodiment, the color buffer is output via either Graphics Interface 117 or Output Controller 180 to a film recording device or written to a peripheral device, e.g., disk drive, tape, compact disk, or the like. 21:54–22:3. If, in step 1125 Graphics Processing Pipeline 103 determines another pass through Graphics Processing Pipeline 103 will not be completed, then in step 1131, Output Controller 180 reads the first or second color buffer, whichever was written during the last pass, for output to Output 185, such as a display or the like. In an alternative embodiment, the color buffer is output via either Graphics Interface 117 or Output Controller 180 to a film recording device or written to a peripheral device, e.g., disk drive, tape, compact disk, or the like. 24:32–41. In step 1128 Raster Operation Unit 165 writes the blended color values to an output buffer (color buffer). The output buffer may be one of the color buffers read in step 1124. In step 1132 Output Controller 180 reads the output buffer for output to Output 185, such as a display or the like. In an alternative embodiment, the output buffer is output via either Graphics Interface 117 or Output Controller 180 to a film recording device or written to a peripheral device, e.g., disk drive, tape, compact disk, or the like.

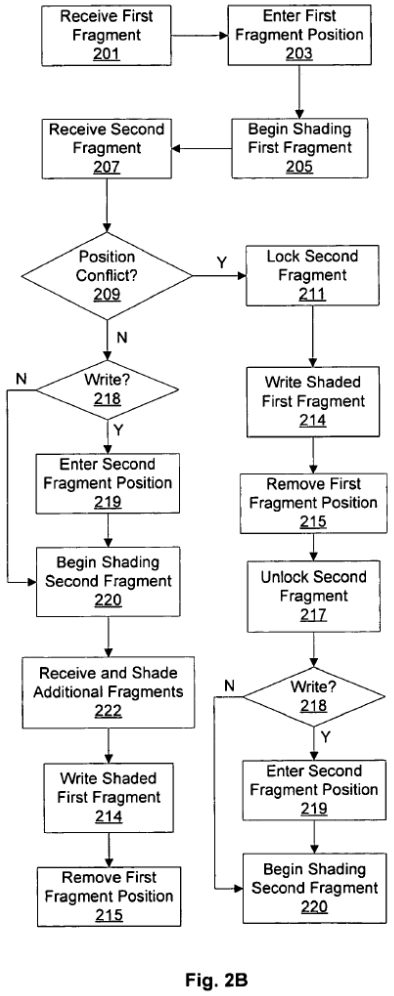
U.S. Patent No. 8,648,867	Kirk
	FIG. 12A is an alternate embodiment of Computing System 100 in accordance with one or more aspects of the present invention. In this embodiment Programmable Graphics Processor 105 includes, among other components, a Front End 1230 that receives commands from Host Computer 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. 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. 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

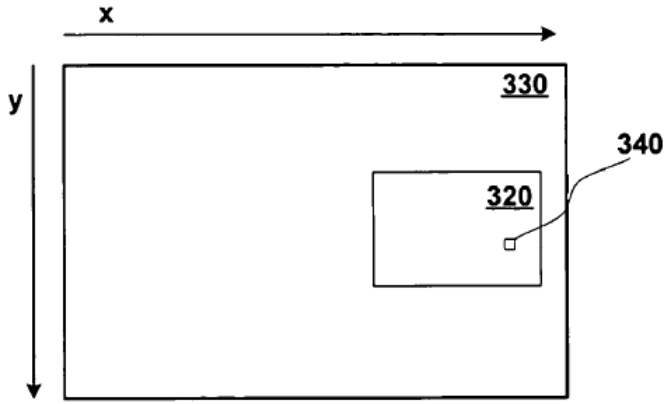
U.S. Patent No. 8,648,867	Kirk
	such that data processing operations are performed in multiple passes through those units or in multiple passes within Programmable Graphics Processing Pipeline 1250. In one embodiment Programmable Graphics Processing Pipeline 1250 performs geometry computations, rasterization, and pixel computations. Therefore, Programmable Graphics Processing Pipeline 1250 is programmed to operate on surface, primitive, vertex, fragment, pixel, sample, or any other data. 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. 26:10–27:11. 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 '867 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 A1–A13, and Appendix A.
16[c]: performing, by at least one graphics processing unit during the first computational cycle, at least	Kirk discloses "performing, by at least one graphics processing unit during the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data." *See e.g.*:

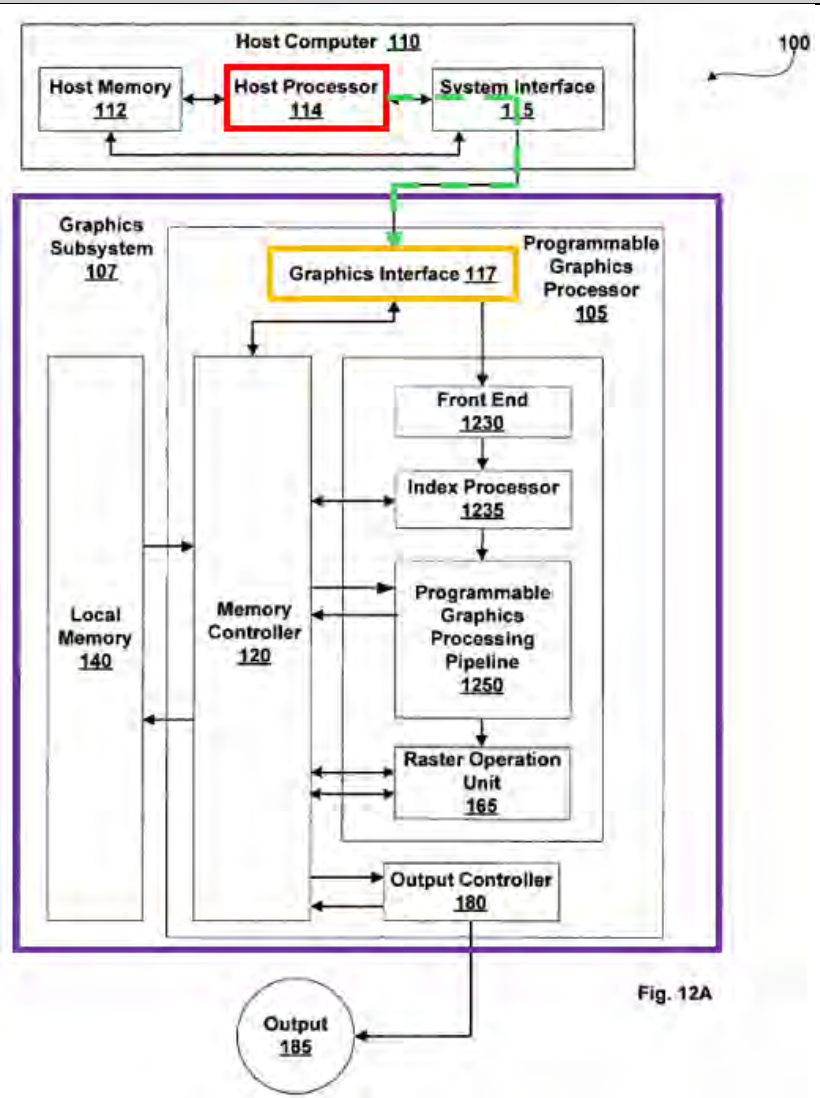
U.S. Patent No. 8,648,867	Kirk
one calculation on the first portion of the input data as to generate first output data;	*As a non-limiting example, Kirk discloses at least one graphics processing units (e.g., Execution Pipelines 1240) that performs at least one calculation (e.g., one or more steps of the 'numerical simulation' described in 16[pre]) on the first portion of the input data (see 16[a]) as to generate first output data (e.g., output data and/or processed samples).*

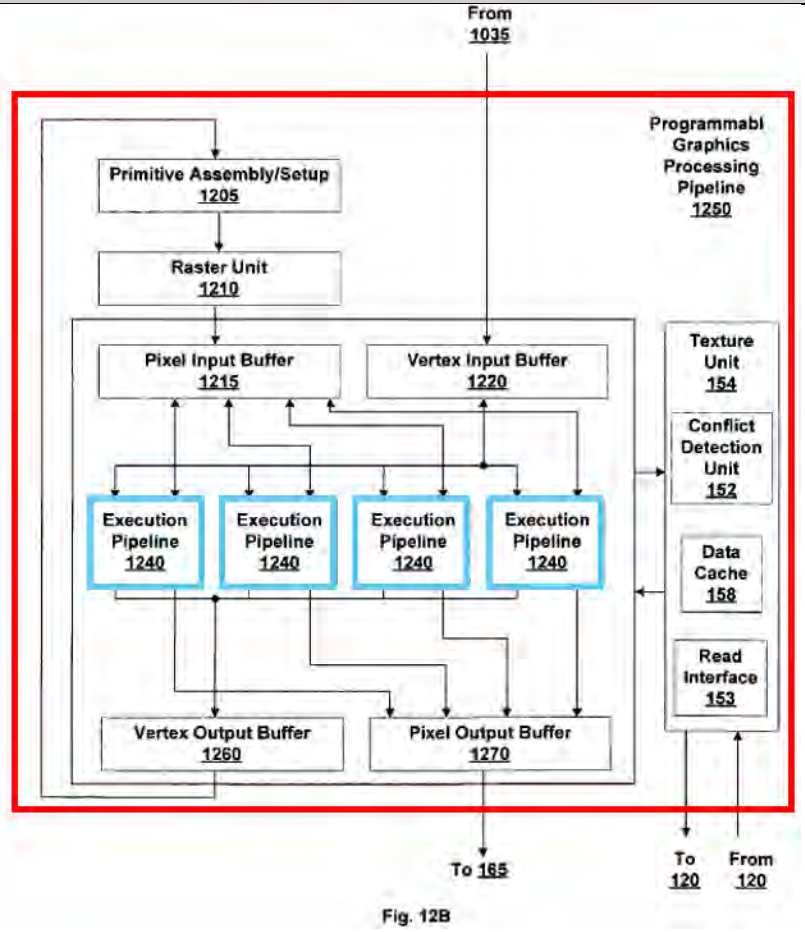
U.S. Patent No. 8,648,867	Kirk
	 Fig. 1B Fig. 1B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 2A Fig. 2A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 2B Fig. 2B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 3A Fig. 3A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated).

U.S. Patent No. 8,648,867	Kirk
	 Fig 12B (annotated). 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

U.S. Patent No. 8,648,867	Kirk
	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. 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. 3:32–67. 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

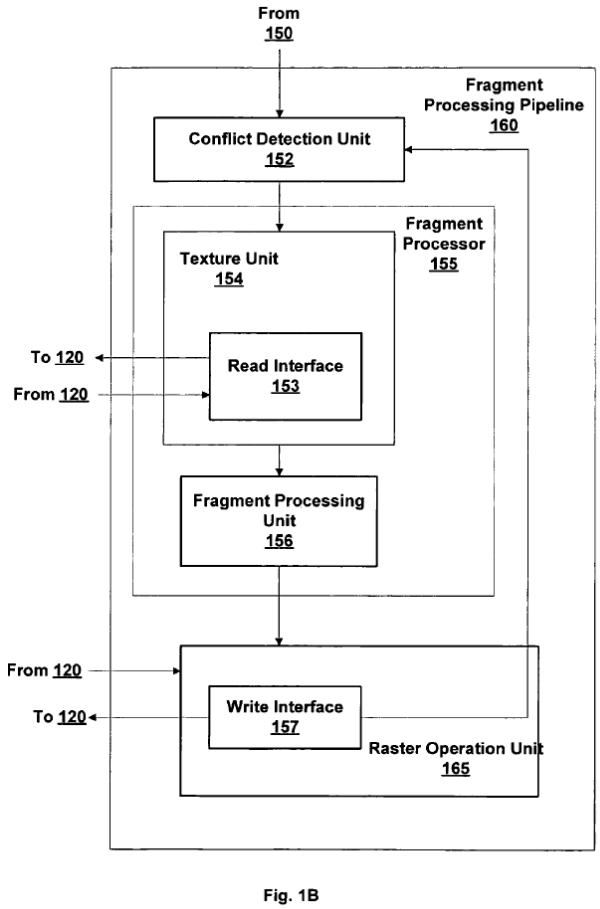
U.S. Patent No. 8,648,867	Kirk
	program instructions. A Texture Unit 154, within Fragment Processor 155, receives the fragment data and fragment program instructions output by Conflict Detection Unit 152. A Read Interface 153, within Texture Unit 154, reads additional fragment program instructions and 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. 6:4–33. 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. 7:7–29. 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

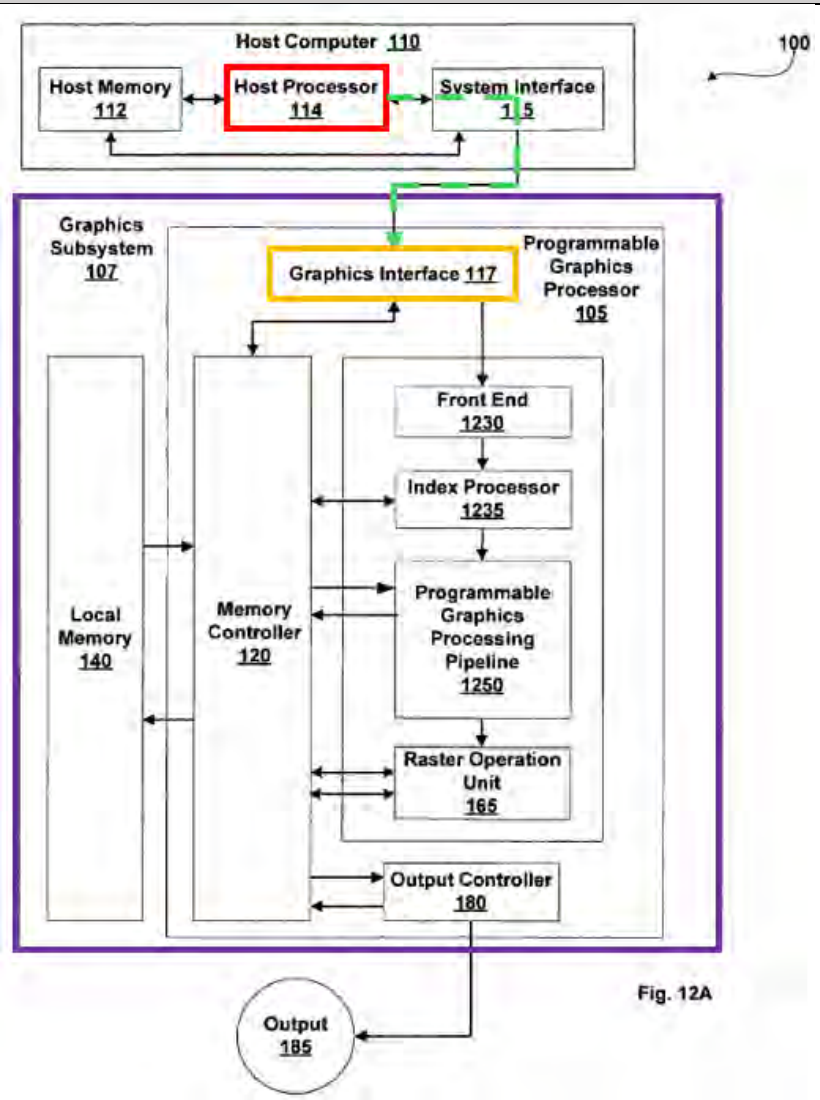
U.S. Patent No. 8,648,867	Kirk
	may be converted into coordinates within a display, e.g., screen coordinates, by applying coordinate offsets based on a position of the buffer within the display. In another embodiment the position is represented as an address for a location in graphics memory. In yet 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. 7:55–8:16. 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. In step 1123, Fragment Processor 155 determines if another fragment will be processed, and, if so, returns to step 1101. In step 1101 Fragment Processor 155 is configured by fragment program instructions to compute the fragment depth value of another fragment for another position. In step 1103 Texture Unit 154 reads a depth buffer value for the other position. The read is performed using the PLD instruction, and the depth buffer value read is stored in Register 159. Because Fragment Processor 155 is pipelined, processing of the other fragment may begin before processing of the fragment is completed. Therefore, the PLD instruction should be used to detect and avoid RAW position conflicts. When the PLD instruction is not used, Fragment Processing Pipeline 160 may be flushed using an instruction to assure processing of the other fragment does

U.S. Patent No. 8,648,867	Kirk
	not begin until the processed fragment is output to graphics memory. Flushing should be minimized to effectively utilize computing resources within Fragment Processing Pipeline 160 and to maximize fragment throughput. 21:37–55. 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 instruc-tions 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. 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. 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.

U.S. Patent No. 8,648,867	Kirk
	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 Pipe-lines 1240. A sample is accepted by an Execution Pipeline 1240 when a processing thread within the Execution Pipeline 1240 is available. 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. 26:24–27:28; *see also* 27:29–46. 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 '867 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

U.S. Patent No. 8,648,867	Kirk
	explained in these invalidity contentions, including the cover pleading and charts A1–A13, and Appendix A.
16[d]: storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory; and	Kirk discloses "storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory." *See e.g.*: *See* 16[b]. *As a non-limiting example, Kirk discloses storage at the direction and/or under the control of the accelerator controller (e.g., Graphics Interface 117, Front End 1230, and Index Processor 1235, and/or Memory Controller 120) of first output data into a "second partition" (e.g., a region including the location and stored data in Vertex Output Buffer 1260 and/or Texture Unit 154). Kirk also discloses referencing partitions by pointers (e.g., accessing data using an address, which can be stored in a pointer, for a first location in graphics memory).*

U.S. Patent No. 8,648,867	Kirk
	 Fig. 1B Fig. 1B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated).

U.S. Patent No. 8,648,867	Kirk
	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. 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.

U.S. Patent No. 8,648,867	Kirk
	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 5 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 10 associated with the output data. 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. 4:57–5:32. 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

U.S. Patent No. 8,648,867	Kirk
	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.

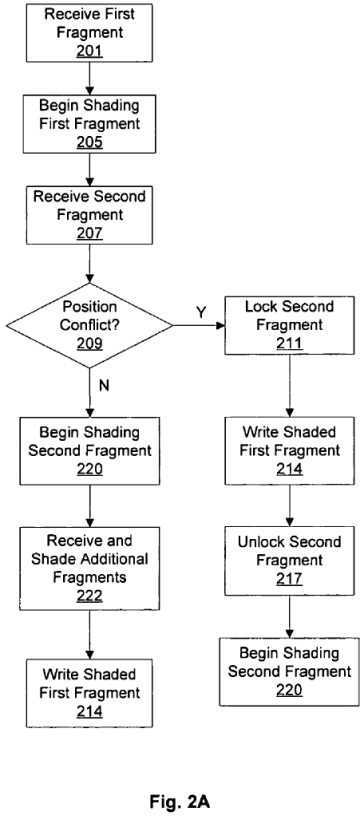
U.S. Patent No. 8,648,867	Kirk
	In step 209 Conflict Detection Unit 152 determines if a RAW position conflict exists for the position associated with the second fragment, and, if so, in step 211 Conflict Detection Unit 152 locks processing of the second fragment. Locking a fragment prevents any processing 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. 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. 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

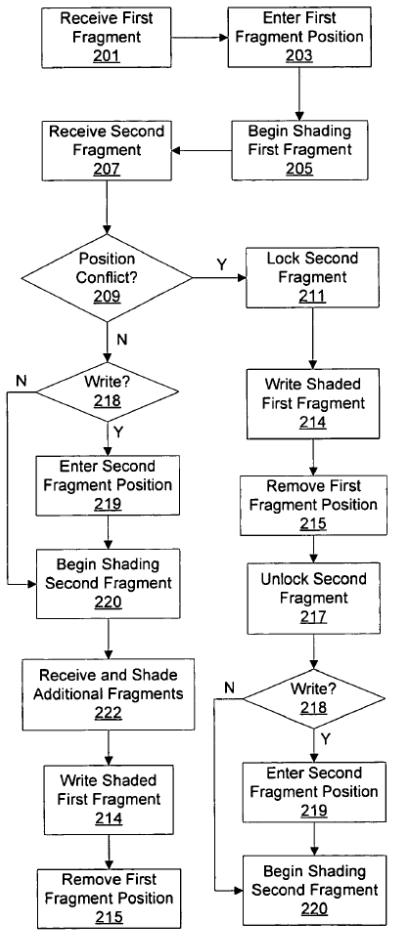
U.S. Patent No. 8,648,867	Kirk
	normalized device coordinates (NDC) space. Each Execution Pipeline 1240 communicates with Texture Unit 154 using Read Interface 153 to read program instructions and graphics data stored in buffers in graphics memory via Memory Controller 120. An optional Data Cache 158 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. 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 35 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. 27:12–39. 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. *Additionally and alternatively, as a non-limiting example, the Memory Controller 120 is part of the "accelerator controller," and storing of first output data occurs "by the accelerator controller" because access to Local Memory 140 occurs "via Memory Controller 120."* 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

U.S. Patent No. 8,648,867	Kirk
	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. 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. 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

U.S. Patent No. 8,648,867	Kirk
	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. 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

U.S. Patent No. 8,648,867	Kirk
	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. 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 '867 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 A1–A13, and Appendix A.
16[e]: swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation.	Kirk discloses "swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation." *See e.g.*: *As a non-limiting example, Kirk discloses writing output from one step to a buffer, then reading that same buffer as input during a subsequent step. When using pointers, as disclosed in Kirk, this is efficiently performed by swapping the first pointer to output data with the second pointer to input data.*

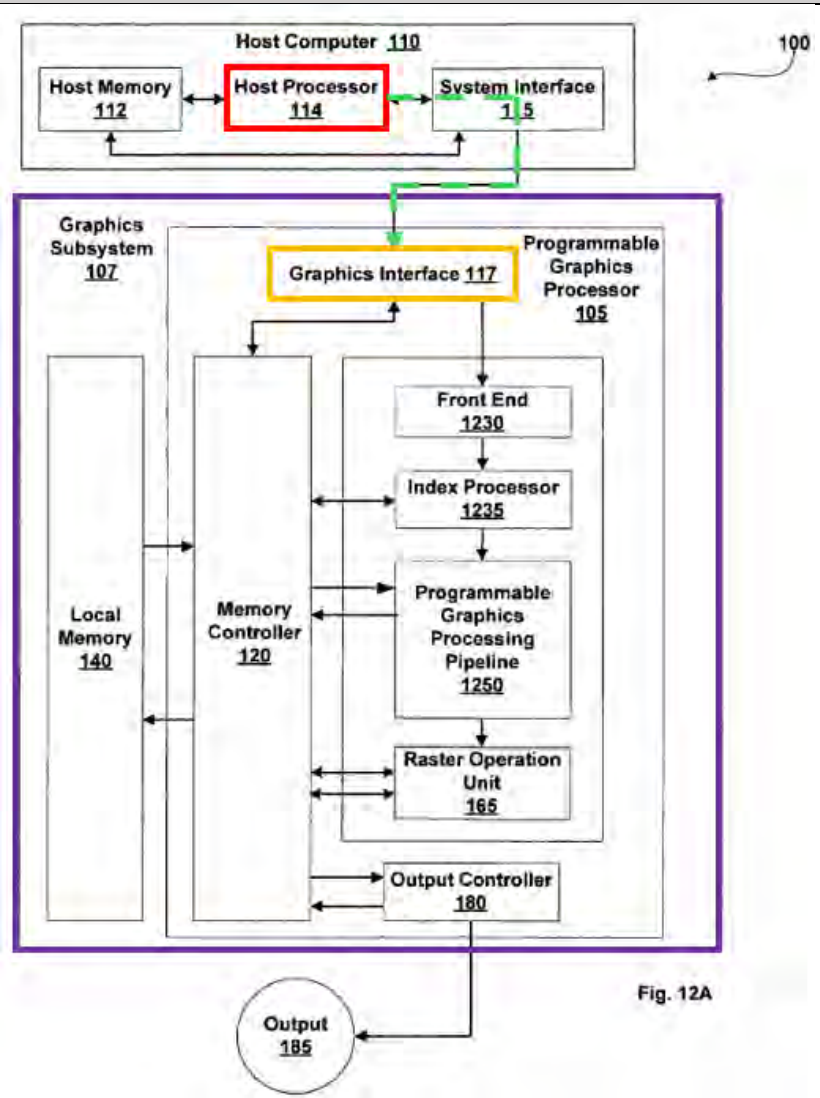
U.S. Patent No. 8,648,867	Kirk
	 Fig. 2A Fig. 2A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 2B Fig. 2B. 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

U.S. Patent No. 8,648,867	Kirk
	location corresponding to the source address is read after the write to the source address is complete. 1:48–54. 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. 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 Graph- ics 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 Out- put 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.

U.S. Patent No. 8,648,867	Kirk
	4:27–67. 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 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. 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 '867 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 A1–A13, and Appendix A.

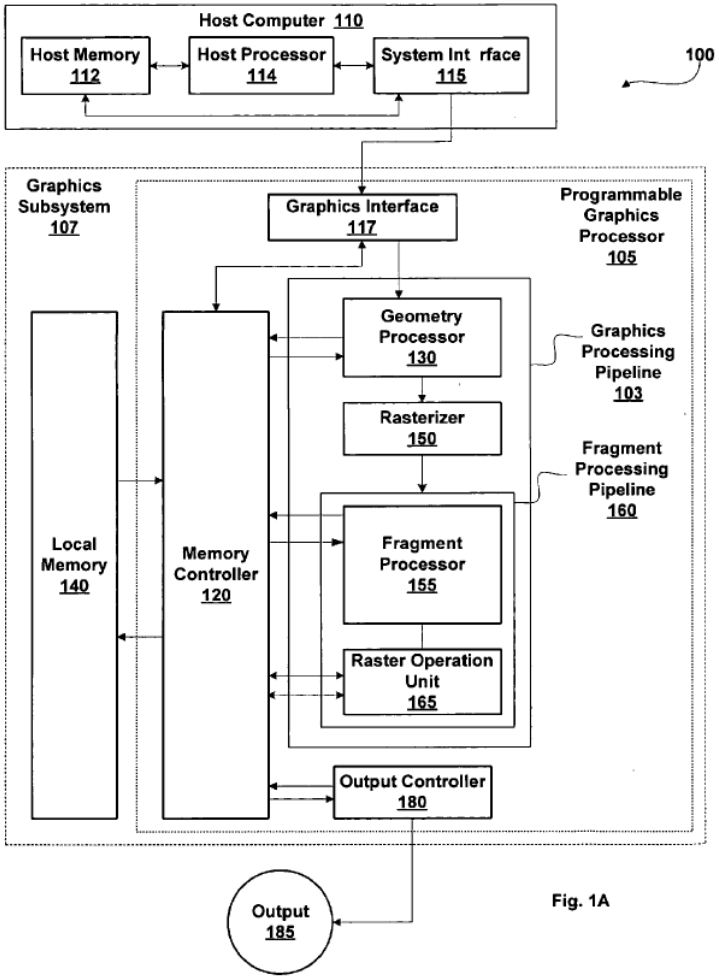
U.S. Patent No. 8,648,867	Kirk
17: The method as claimed in claim 16, further comprising: sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit.	Kirk discloses "sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit." *See e.g.*: *As a non-limiting example, Kirk discloses sending, by the accelerator controller (e.g., at the direction and control of Graphics Interface 117, Front End 1230, and Index Processor 1235, and/or Memory Controller 120), instructions for performing the at least one calculation (e.g., program instructions) to the at least one processing unit.*

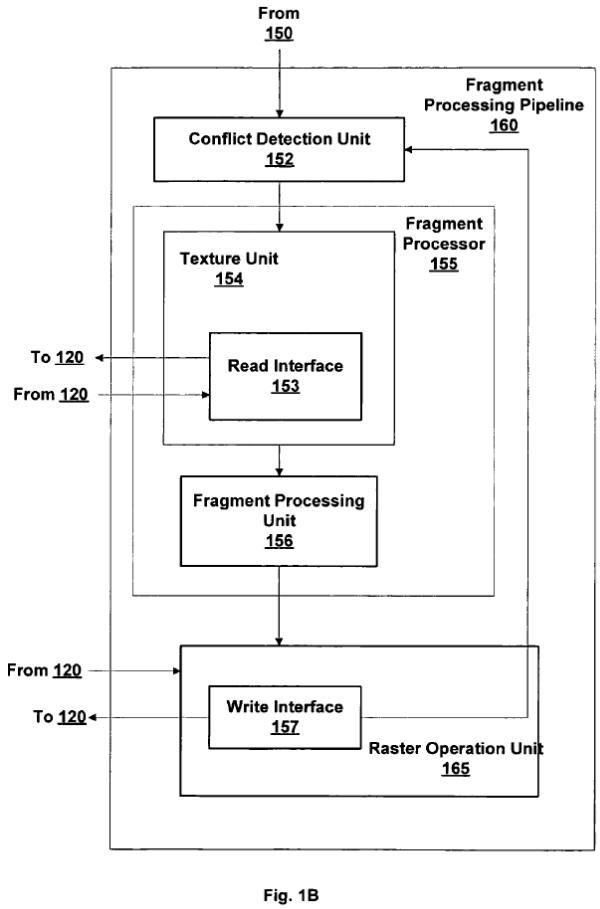
U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated).

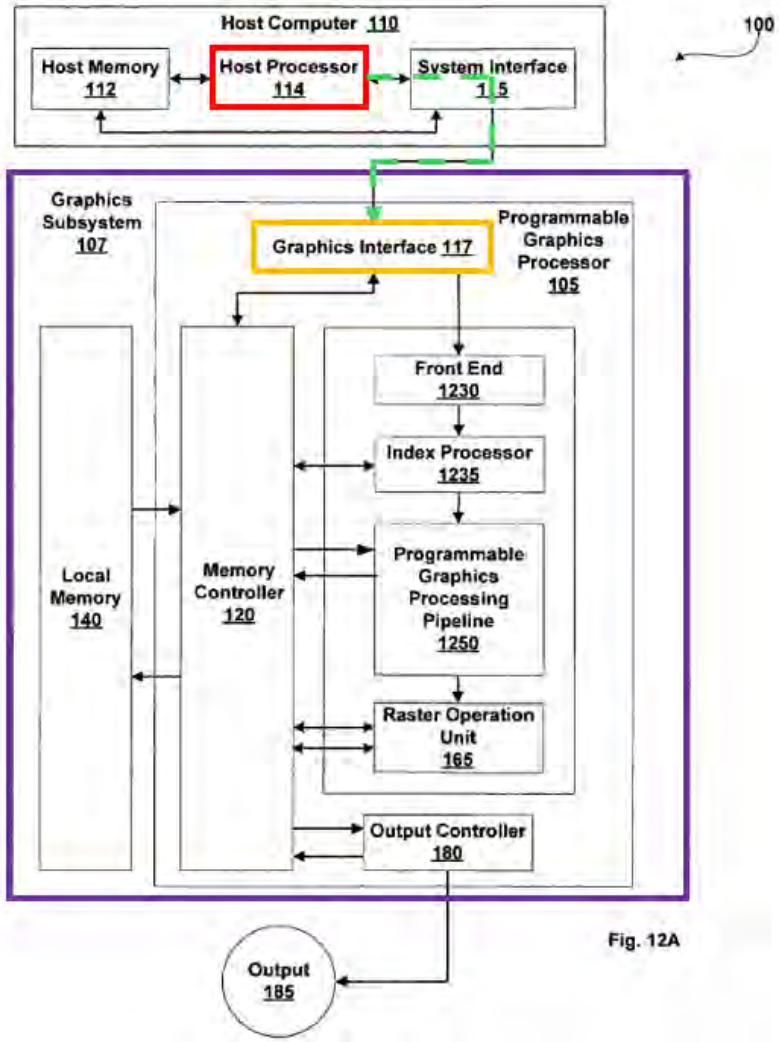
U.S. Patent No. 8,648,867	Kirk
	Various embodiments of a method of the invention include processing fragment program instructions. A pixel load instruction including a source address corresponding to a location within the buffer is received. A write to the source address is determined to be pending. Data stored in the location corresponding to the source address is read after the write to the source address is complete. 1:48–54. 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. 1:63–2:4. 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. 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

U.S. Patent No. 8,648,867	Kirk
	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; *see also* 3:32–6:55. 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. 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 '867 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

U.S. Patent No. 8,648,867	Kirk
	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 A1–A13, and Appendix A.
18: The method as claimed in claim 16, further comprising: partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation.	Kirk discloses "partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation." *See e.g.*: *See* 16[b], 16[d]. *As a non-limiting example, Kirk discloses that its accelerator memory (e.g., e.g., a region including the location and stored data in Vertex Output Buffer 1260 and/or Texture Unit 154) is partitioned in a first partition to store input data and a second partition to store output data.* 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 '867 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 A1–A13, and Appendix A.
19: The method of claim 16, further comprising: transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle.	Kirk discloses "transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle." *See e.g.*: *As a non-limiting example, Kirk discloses transferring, by the accelerator controller (e.g., at the direction and control of Graphics Interface 117, Front End 1230, and Index Processor 1235, and/or Memory Controller 120), the first output data to the main memory (e.g., the Host Memory 112) during the second computational cycle (e.g., after the write to the source address is complete).*

U.S. Patent No. 8,648,867	Kirk
	 Fig. 1A.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 1B Fig. 1B.

U.S. Patent No. 8,648,867	Kirk
	 Fig. 12A (annotated). 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

U.S. Patent No. 8,648,867	Kirk
	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. FIG. lA 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. 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. 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)

U.S. Patent No. 8,648,867	Kirk
	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. 3:17–67. 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. 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

U.S. Patent No. 8,648,867	Kirk
	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 5 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. 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,

U.S. Patent No. 8,648,867	Kirk
	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. 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 Detec- tion 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 com- plete. 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. 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

U.S. Patent No. 8,648,867	Kirk
	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. 11:22–40. 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. 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 '867 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 A1–A13, and Appendix A.