(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 March 2018 (29.03.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/057294 A1

(51) International Patent Classification:
G06T 15/00 (2011.01)

(21) International Application Number:
PCT/US2017/050345

(22) International Filing Date:
06 September 2017 (06.09.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

62/398,211	22 September 2016 (22.09.2016)	US
15/389,481	23 December 2016 (23.12.2016)	US
17150441.8	05 January 2017 (05.01.2017)	EP

(71) Applicant: **ADVANCED MICRO DEVICES, INC.** [US/US]; One AMD Place, Sunnyvale, California 94085 (US).

(72) Inventors: **NIJASURE, Mangesh P.**; c/o Advanced Micro Devices, Inc., 3501 Quadrangle Blvd., Suite 375, Orlando, Florida 32817 (US). **RAMSEY, Randy W.**; c/o Advanced Micro Devices, Inc., 3501 Quadrangle Blvd., Suite 375, Orlando, Florida 32817 (US). **MARTIN, Todd**; c/o Advanced Micro Devices, Inc., 3501 Quadrangle Blvd., Suite 375, Orlando, Florida 32817 (US).

(74) Agent: **HAMBERGER, Joshua A.**; 30 S. 17th Street, United Plaza, Suite 1800, Philadelphia, Pennsylvania 19103 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA,

(54) Title: COMBINED WORLD-SPACE PIPELINE SHADER STAGES

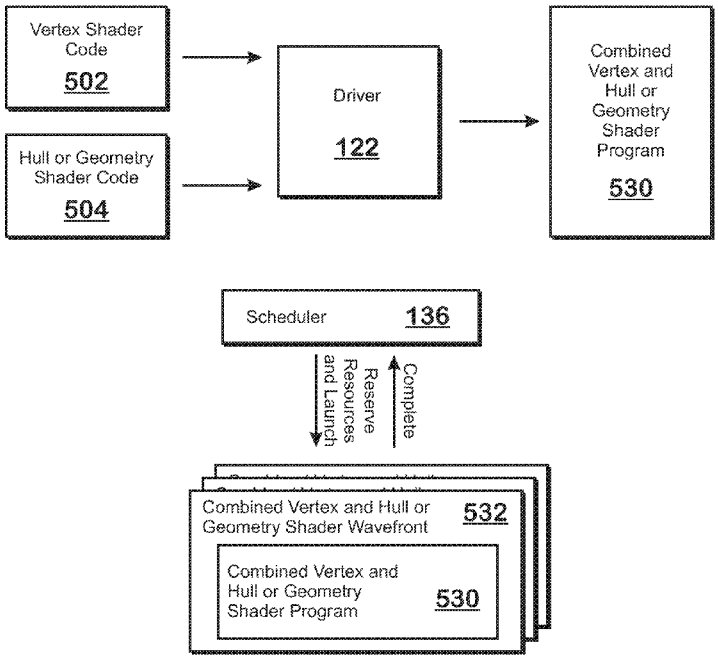


FIG. 5B

(57) Abstract: Improvements to graphics processing pipelines are disclosed. More specifically, the vertex shader stage, which performs vertex transformations, and the hull or geometry shader stages, are combined. If tessellation is disabled and geometry shading is enabled, then the graphics processing pipeline includes a combined vertex and graphics shader stage. If tessellation is enabled, then the graphics processing pipeline includes a combined vertex and hull shader stage. If tessellation and geometry shading are both disabled, then the graphics processing pipeline does not use a combined shader stage. The combined shader stages improve efficiency by reducing the number of executing instances of shader programs and associated resources reserved.

WO 2018/057294 A1

[Continued on next page]

SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

COMBINED WORLD-SPACE PIPELINE SHADER STAGES

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 62/398,211 filed September 22, 2016, U.S. Patent Application No. 15/389,481 filed December 23, 2016 and European Application No. EP17150441.8 filed January 5, 2017, which are incorporated by reference as if fully set forth herein.

TECHNICAL FIELD

[0002] The disclosed embodiments are generally directed to graphics processing pipelines, and in particular, to combined world-space pipeline shader stages.

BACKGROUND

[0003] Three-dimensional graphics processing pipelines accept commands from a host (such as a central processing unit of a computing system) and process those commands to generate pixels for display on a display device. Graphics processing pipelines include a number of stages that perform individual tasks, such as transforming vertex positions and attributes, calculating pixel colors, and the like. Graphics processing pipelines are constantly being developed and improved.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] A more detailed understanding may be had from the following description, given by way of example in conjunction with the accompanying drawings wherein:

[0005] Figure 1 is a block diagram of an example device in which one or more disclosed embodiments may be implemented;

[0006] Figure 2 is a block diagram of the device of Figure 1, illustrating additional detail;

[0007] Figure 3 is a block diagram showing additional details of the graphics processing pipeline illustrated in Figure 2;

[0008] Figures 4A and 4B illustrate configurations for the graphics processing pipeline, according to examples;

[0009] Figures 5A and 5B illustrate aspects of combined shader stages involving the driver and the scheduler illustrated in Figure 2, according to examples;

[0010] Figure 6 illustrates operations for enabling or disabling wavefronts for the combined vertex and hull or geometry shader stage in order to accommodate that change in workload at the boundary between shader stages, according to an example; and

[0011] Figure 7 is a flow diagram of a method for executing a combined vertex and hull or geometry shader program for a combined vertex and hull or geometry shader stage, according to an example

DETAILED DESCRIPTION

[0012] The present disclosure is directed to improvements in the graphics processing pipeline. More specifically, the vertex shader stage, which performs vertex transformations, and the hull or geometry shader stages, are combined. If tessellation is disabled and geometry shading is enabled, then the graphics processing pipeline includes a combined vertex and graphics shader stage. If tessellation is enabled, then the graphics processing pipeline includes a combined vertex and hull shader stage. If tessellation and geometry shading are both disabled, then the graphics processing pipeline does not use a combined shader stage. The combined shader stages improve efficiency by reducing the number of executing instances of shader programs and associated resources reserved.

[0013] Figure 1 is a block diagram of an example device 100 in which one or more aspects of the present disclosure are implemented. The device 100 includes, for example, a computer, a gaming device, a handheld device, a set-top box, a television, a mobile phone, or a tablet computer. The device 100 includes a processor 102, a memory 104, a storage device 106, one or more input devices 108, and one or more output devices 110. The device 100 also optionally includes

an input driver 112 and an output driver 114. It is understood that the device 100 may include additional components not shown in Figure 1.

[0014] The processor 102 includes a central processing unit (CPU), a graphics processing unit (GPU), a CPU and GPU located on the same die, or one or more processor cores, wherein each processor core may be a CPU or a GPU. The memory 104 is located on the same die as the processor 102, or may be located separately from the processor 102. The memory 104 includes a volatile or non-volatile memory, for example, random access memory (RAM), dynamic RAM, or a cache.

[0015] The storage device 106 includes a fixed or removable storage, for example, a hard disk drive, a solid state drive, an optical disk, or a flash drive. The input devices 108 include a keyboard, a keypad, a touch screen, a touch pad, a detector, a microphone, an accelerometer, a gyroscope, a biometric scanner, or a network connection (e.g., a wireless local area network card for transmission and/or reception of wireless IEEE 802 signals). The output devices 110 include a display, a speaker, a printer, a haptic feedback device, one or more lights, an antenna, or a network connection (e.g., a wireless local area network card for transmission and/or reception of wireless IEEE 802 signals).

[0016] The input driver 112 communicates with the processor 102 and the input devices 108, and permits the processor 102 to receive input from the input devices 108. The output driver 114 communicates with the processor 102 and the output devices 110, and permits the processor 102 to send output to the output devices 110. The output driver 114 includes an accelerated processing device (APD) 116 which is coupled to a display device 118. The APD is configured to accept compute commands and graphics rendering commands from processor 102, to process those compute and graphics rendering commands, and to provide pixel output to display device 118 for display.

[0017] The APD 116 includes one or more parallel processing units configured to perform computations in accordance with a single-instruction-multiple-data ("SIMD") paradigm. However, functionality described as being

performed by the APD 116 may also be performed by processing devices that do not process data in accordance with a SIMD paradigm.

[0018] Figure 2 is a block diagram of the device 100, illustrating additional details related to execution of processing tasks on the APD 116. The processor 102 maintains, in system memory 104, one or more control logic modules for execution by the processor 102. The control logic modules include an operating system 120, a driver 122, and applications 126, and may optionally include other modules not shown. These control logic modules control various aspects of the operation of the processor 102 and the APD 116. For example, the operating system 120 directly communicates with hardware and provides an interface to the hardware for other software executing on the processor 102. The driver 122 controls operation of the APD 116 by, for example, providing an application programming interface ("API") to software (e.g., applications 126) executing on the processor 102 to access various functionality of the APD 116. The driver 122 also includes a just-in-time compiler that compiles shader code into shader programs for execution by processing components (such as the SIMD units 138 discussed in further detail below) of the APD 116.

[0019] The APD 116 executes commands and programs for selected functions, such as graphics operations and non-graphics operations, which may be suited for parallel processing. The APD 116 can be used for executing graphics pipeline operations such as pixel operations, geometric computations, and rendering an image to display device 118 based on commands received from the processor 102. The APD 116 also executes compute processing operations that are not directly related to graphics operations, such as operations related to video, physics simulations, computational fluid dynamics, or other tasks, based on commands received from the processor 102 or that are not part of the "normal" information flow of a graphics processing pipeline.

[0020] The APD 116 includes shader engines 132 (which may collectively be referred to herein as "programmable processing units 202") that include one or more SIMD units 138 that are configured to perform operations at the request of the processor 102 in a parallel manner according to a SIMD paradigm. The

SIMD paradigm is one in which multiple processing elements share a single program control flow unit and program counter and thus execute the same program but are able to execute that program with different data. In one example, each SIMD unit 138 includes sixteen lanes, where each lane executes the same instruction at the same time as the other lanes in the SIMD unit 138 but can execute that instruction with different data. Lanes can be switched off with predication if not all lanes need to execute a given instruction. Predication can also be used to execute programs with divergent control flow. More specifically, for programs with conditional branches or other instructions where control flow is based on calculations performed by individual lanes, predication of lanes corresponding to control flow paths not currently being executed, and serial execution of different control flow paths, allows for arbitrary control flow to be followed.

[0021] The basic unit of execution in shader engines 132 is a work-item. Each work-item represents a single instantiation of a shader program that is to be executed in parallel in a particular lane. Work-items can be executed simultaneously as a "wavefront" on a single SIMD unit 138. Multiple wavefronts may be included in a "work group," which includes a collection of work-items designated to execute the same program. A work group can be executed by executing each of the wavefronts that make up the work group. The wavefronts may be executed sequentially on a single SIMD unit 138 or partially or fully in parallel on different SIMD units 138. Wavefronts can be thought of as instances of parallel execution of a shader program, where each wavefront includes multiple work-items that execute simultaneously on a single SIMD unit 138 in line with the SIMD paradigm (e.g., one instruction control unit executing the same stream of instructions with multiple data).

[0022] SIMD units 138 store working data in vector registers 206, which are configured to store different values for different work-items executing the same instruction in the SIMD units 138 or in scalar registers 208 which are configured to store single values for use, for example, when an instruction in a shader program uses the same operand value for each work-item. A local data

store memory 212 in each shader engine 132 also stores values for use by shader programs. The local data store memory 212 may be used for data that cannot fit into the vector registers 206 or scalar registers 208 but which is used by the shader programs. The physical proximity of the local data store memory 212 provides improved latency as compared with other memories such as memory 210 in the APD 116 that is not included within shader engines 132 or memory 104 that is not within the APD 116.

[0023] A scheduler 136 is configured to perform operations related to scheduling various wavefronts on different shader engines 132 and SIMD units 138. Wavefront bookkeeping 204 inside scheduler 136 stores data for pending wavefronts, which are wavefronts that have launched and are either executing or "asleep" (e.g., waiting to execute or not currently executing for some other reason). In addition to identifiers identifying pending wavefronts, wavefront bookkeeping 204 also stores indications of resources used by each wavefront, including registers such as vector registers 206 and/or scalar registers 208, portions of a local data store memory 212 assigned to a wavefront, portions of a memory 210 not local to any particular shader engine 132, or other resources assigned to the wavefront.

[0024] The parallelism afforded by the shader engines 132 is suitable for graphics related operations such as pixel value calculations, vertex transformations, tessellation, geometry shading operations, and other graphics operations. A graphics processing pipeline 134 which accepts graphics processing commands from the processor 102 thus provides computation tasks to the shader engines 132 for execution in parallel.

[0025] The shader engines 132 are also used to perform computation tasks not related to graphics or not performed as part of the "normal" operation of a graphics processing pipeline 134 (e.g., custom operations performed to supplement processing performed for operation of the graphics processing pipeline 134). An application 126 or other software executing on the processor 102 transmits programs (often referred to as "compute shader programs," which

may be compiled by the driver 122) that define such computation tasks to the APD 116 for execution.

[0026] Figure 3 is a block diagram showing additional details of the graphics processing pipeline 134 illustrated in Figure 2. The graphics processing pipeline 134 includes stages that each performs specific functionality. The stages represent subdivisions of functionality of the graphics processing pipeline 134. Each stage is implemented partially or fully as shader programs executing in the programmable processing units 202, or partially or fully as fixed-function, non-programmable hardware external to the programmable processing units 202.

[0027] The input assembler stage 302 reads primitive data from user-filled buffers (e.g., buffers filled at the request of software executed by the processor 102, such as an application 126) and assembles the data into primitives for use by the remainder of the pipeline. The input assembler stage 302 can generate different types of primitives based on the primitive data included in the user-filled buffers. The input assembler stage 302 formats the assembled primitives for use by the rest of the pipeline.

[0028] The vertex shader stage 304 processes vertices of the primitives assembled by the input assembler stage 302. The vertex shader stage 304 performs various per-vertex operations such as transformations, skinning, morphing, and per-vertex lighting. Transformation operations may include various operations to transform the coordinates of the vertices. These operations may include one or more of modeling transformations, viewing transformations, projection transformations, perspective division, and viewport transformations. Herein, such transforms are considered to modify the coordinates or "position" of the vertices on which the transforms are performed. Other operations of the vertex shader stage 304 that modify attributes other than the coordinates are considered to modify non-position attributes.

[0029] The vertex shader stage 304 is implemented partially or fully as vertex shader programs to be executed on one or more shader engines 132. The vertex shader programs are provided by the processor 102 as programs that are

pre-written by a computer programmer. The driver 122 compiles such computer programs to generate the vertex shader programs having a format suitable for execution within the shader engines 132.

[0030] The hull shader stage 306, tessellator stage 308, and domain shader stage 310 work together to implement tessellation, which converts simple primitives into more complex primitives by subdividing the primitives. The hull shader stage 306 generates a patch for the tessellation based on an input primitive defined by a set of vertices and other information. The tessellator stage 308 generates a set of samples (which may include vertices specified by barycentric coordinates) for the patch. The domain shader stage 310 calculates vertex positions for the vertices corresponding to the samples for the patch (by, for example, converting the barycentric coordinates to world-space coordinates). The hull shader stage 306 and domain shader stage 310 can be implemented as shader programs to be executed on the programmable processing units 202.

[0031] The geometry shader stage 312 performs vertex operations on a primitive-by-primitive basis. Geometry shader programs typically accept whole primitives (e.g., a collection of vertices) as input and perform operations on those whole primitives as specified by the instructions of the geometry shader programs. A variety of different types of operations can be performed by the geometry shader stage 312, including operations such as point sprite expansion, dynamic particle system operations, fur-fin generation, shadow volume generation, single pass render-to-cubemap, per-primitive material swapping, and per-primitive material setup. Operations for the geometry shader stage 312 may be performed by a shader program that executes on the programmable processing units 202.

[0032] The rasterizer stage 314 accepts and rasterizes simple primitives generated upstream. Rasterization consists of determining which screen pixels (or sub-pixel samples) are covered by a particular primitive. Rasterization is performed by fixed function hardware or may be performed by shader programs executing in the programmable processing units 202.

[0033] The pixel shader stage 316 calculates output values (e.g., color values) for screen pixels based on the primitives generated upstream and the results of rasterization. The pixel shader stage 316 may apply textures from texture memory. Operations for the pixel shader stage 316 are performed by a shader program that executes on the programmable processing units 202.

[0034] The output merger stage 318 accepts output from the pixel shader stage 316 and merges those outputs, performing operations such as z-testing and alpha blending to determine the final color for a screen pixel, which are written to a frame buffer for output to the display device 118.

[0035] As described above, many of the stages illustrated in Figure 3 and described as being included within the graphics processing pipeline 134 can be implemented as shader programs executing within the shader engines 132 illustrated in Figure 2. Various operations occur in the driver 122 and within the APD 116 to facilitate executing shader programs in the shader engines 132.

[0036] One such operation involves facilitating shader input and output data transfer. More specifically, the stages of the graphics processing pipeline 134 typically obtain input data, perform some processing on that input data, and provide output data in response, usually for the next stage of the graphics processing pipeline 134. Shader programs that execute as part of the graphics processing pipeline 134 include instructions or "hints" to the APD 116 that specify inputs and outputs for the shader programs. These hints inform the APD 116 regarding where (e.g., which registers) to place inputs for particular shader programs and where (e.g., which registers) to retrieve outputs from for particular shader programs. This input and output information is used, at least in part, to instruct the APD 116 regarding where to place inputs for a particular shader program and also where to fetch the outputs from for a particular shader program, in order to forward the output data to other parts of the graphics processing pipeline 134 such as fixed function hardware or other shader programs. In one example, a vertex shader program specifies locations (e.g., registers) at which inputs are expected. The APD 116 fetches inputs and places the inputs at those locations. The vertex shader program performs vertex

shading operations on the input vertices, and provides modified vertices as output. The APD 116 fetches these modified vertices and places those vertices at the locations (e.g., registers) specified as inputs by the next stage of the graphics processing pipeline 134.

[0037] Another operation for facilitating execution of shader programs involves reserving resources for wavefronts that are to execute a shader program (e.g., entries in wavefront bookkeeping 204, registers to be used by the wavefronts, portions of local data store memory 212, memory 210, and other memory, as well as other resources to be used by wavefronts) prior to launching the wavefronts to execute the shader program. The quantity of resources to be reserved for wavefronts for a particular shader program are based at least partially on the instructions of the shader programs. More specifically, shader programs typically include instructions, each of which can specify particular registers to use as operands. The APD 116 determines a number of registers to reserve for a wavefront based on the registers specified in the instructions of a shader program that the wavefront is to execute. In one example, ten different vector registers are specified by a particular wavefront that is to execute with 64 work items. Thus the APD 116 determines that 10 x 64 = 640 registers need to be reserved to execute that wavefront. Similarly, instructions may specify locations in memory as operands. The APD 116 determines a total amount of memory to reserve for a wavefront based on the memory locations specified by the instructions of the shader program. Other resources required for wavefronts are also reserved based on the characteristics of the shader programs that the wavefronts are to execute.

[0038] Another operation that occurs to prepare wavefronts to execute includes receiving an indication that wavefronts from a prior shader stage has completed execution. More specifically, some shader stages are dependent on other shader stages. For example, if tessellation (which involves the hull shader stage 306, tessellator stage 308, and domain shader stage 310) is enabled, then the hull shader stage 306 is dependent on the results from the vertex shader stage 304 to execute. Thus, wavefronts that are to execute hull shader programs

on a particular set of vertices wait for the wavefronts executing a vertex shader program on those same vertices to complete. This "handoff" is typically facilitated by the scheduler 136, which receives notifications that particular wavefronts are complete and launches wavefronts for subsequent stages in response.

[0039] The above operations illustrate that each stage of the graphics processing pipeline 134 that is implemented at least partially via shader programs is associated with some amount of overhead. For example, different wavefronts are launched for different shader program types (e.g., one type of wavefront is launched to execute a vertex shader program and another type of wavefront is launched to execute a hull shader program). Thus, a larger number of shader stages is generally associated with a greater number of wavefronts that are tracked by the scheduler 136. Other overhead involves overhead related to transferring data between shader stages, and overhead related to the amount of resources that are reserved for each different shader program type. For this reason, combining certain shader stages could help to reduce such overhead and improve performance. Two shader stages that can be combined are the vertex and hull shader stages and the vertex and geometry shader stages.

[0040] Figures 4A and 4B illustrate configurations for the graphics processing pipeline, according to examples. These alternative configurations include the configuration of Figure 4A, in which the vertex shader stage and geometry shader stage 420 are combined into a single shader stage – the combined vertex shader and geometry shader stage 420 – and the configuration of Figure 4B, in which the vertex shader stage and hull shader stage 306 are combined into a single shader stage – the combined vertex shader and hull shader stage 410. Herein, the configuration of the graphics processing pipeline 134 illustrated in Figure 4A is referred to as the "vertex/geometry shader configuration 402" and the configuration of the graphics processing pipeline 134 illustrated in Figure 4B is referred to as the "vertex/hull shader configuration 404."

[0041] Various stages of the graphics processing pipeline 134 can be enabled or disabled. Specifically, geometry shading, which is implemented by the geometry shader stage 312, can be enabled or disabled, and the stages implementing tessellation (the hull shader stage 306, the tessellator stage 308, and the domain shader stage 310) can be enabled or disabled together. If both tesselation and geometry shading are disabled, then neither of the configurations illustrated in Figures 4A and 4B are used. In both Figures 4A and 4B, a disabled stage is indicated with an arrow that flows around the disabled stage to a subsequent stage in the graphics processing pipeline 134. If geometry shading is enabled and tessellation is disabled, then the vertex/geometry shader configuration 402 of Figure 4A is used. If tessellation is enabled (and regardless of whether geometry shading is enabled), then the vertex/hull shader configuration 404 of Figure 4B is used (the geometry shader stage 312 is illustrated with dotted lines, indicating that use of that stage is optional).

[0042] "Combining" two shader stages means that the shader programs for the two shader stages are combined (e.g., by the driver 122) into a single shader program and wavefronts are launched to execute the combined shader program. More specifically, instead of launching wavefronts of a first type to execute vertex shader programs and launching wavefronts of a second type to execute hull or geometry shader programs, the APD 116 launches wavefronts of a single type – a combined vertex and hull or geometry shader type – to execute a combined shader program for the combined pipeline stage. This combining involves several operations and/or modifications to the APD 116 and graphics processing pipeline 134, including modifying inputs and outputs that the shader programs declare, modifying the manner in which wavefronts execute in the APD 116, modifying the manner in which resources are allocated for the different shader programs, and other operations described herein.

[0043] Figures 5A and 5B illustrate aspects of combined shader stages involving the driver 122 and the scheduler 136 illustrated in Figure 2, according to examples. For purposes of comparative illustration, Figure 5A illustrates uncombined shader stages and Figure 5B illustrates combined shader stages.

[0044] In Figure 5A, the driver 122 receives vertex shader code 502 and hull or geometry shader code 504 from hardware or software requesting such shader code to be executed on the APD 116 as part of the vertex shader stage 304 and the hull shader stage 306 or geometry shader stage 312. The hull or geometry shader code 504 represents either geometry shader code or hull shader code.

[0045] In response to receiving the vertex shader code 502 and the hull or geometry shader code 504, the driver 122 compiles the vertex shader code 502 to generate a vertex shader program 506 and compiles the hull or geometry shader code 504 to generate a hull or geometry shader program 508. Techniques for compiling shader programs provided to a driver 122 (by, e.g., and application 126) generally comprise converting programs specified in a high level language such as AMD Intermediate Language to lower level instructions that are more tied to the hardware and that are understood by the APD 116.

[0046] In the APD 116, the scheduler 136 obtains the compiled vertex shader program 506 and hull or geometry shader program 508 executes those shader programs. Part of this execution involves identifying and reserving resources needed for the shader programs. Another part of this execution involves launching wavefronts to execute the shader programs. Yet another part of this execution involves coordinating wavefront execution between different stages of the pipeline (in other words, making sure that wavefronts for a later stage waits for execution of wavefronts of an earlier stage to execute before completing, where execution in the later stage is dependent on results of the earlier stage).

[0047] These aspects of execution are illustrated in Figure 5A. The scheduler 136 reserves resources and launches vertex shader wavefronts that execute vertex shader programs. The scheduler 136 then waits for "complete" signals for the outstanding vertex shader wavefronts 510. Upon receiving the "complete" signals, the scheduler 136 reserves resources for and launches the hull or geometry shader wavefronts 512 to execute the hull or geometry shader

programs 508. As described above, resources include registers, memory, tracking resources for tracking execution of the wavefronts, and other resources.

[0048] Figure 5B illustrates the combined shader stages. For these combined shader stages, the driver 122 receives the vertex shader code 502 and hull or geometry shader code 504 (from, e.g., an application 126) and compiles these different portions of code into a single combined vertex and hull or geometry shader program 530. The hull or geometry shader code 504 represents either geometry shader code for a configuration of the graphics processing pipeline 134 in which tessellation is disabled but geometry shading is enabled (Figure 4A) or hull shader code for a configuration of the graphics processing pipeline 134 in which tessellation is disabled (Figure 4B).

[0049] To combine shader stages, the driver 122 compiles the vertex shader code 502 and hull or geometry shader code 504 to generate compiled shader programs and "stitches" the compiled shader programs together. Stitching the two shader programs together means that the two shader programs are concatenated and then the combination is modified as appropriate.

[0050] One way in which the concatenated shader programs are modified relates to the inputs and outputs defined for the shader programs. More specifically, each shader program defines inputs and outputs to the shader programs. These defined inputs and outputs act as hints to the APD 116. When a wavefront is to begin execution of a shader program, the APD 116 ensures that the values indicated as inputs are placed in locations (e.g., registers) specified by the shader program. When a wavefront completes execution, the APD 116 retrieves the data from the locations (e.g., registers) indicated as storing the outputs. The APD 116 may copy that data to other locations (e.g., registers) for shader programs in subsequent shader stages that use the output data. For example, for the graphics processing pipeline 134 illustrated in Figure 3, the APD 116 copies shaded vertices from locations specified as outputs in vertex shader programs executed in the vertex shader stage 304 to locations specified as inputs for hull shader programs to execute in the hull shader stage 306.

[0051] The vertex shader code 502 defines outputs and the hull or geometry shader code 504 defines inputs. However, because the combined vertex and hull or geometry shader program 530 is a single shader program, instead of two separate shader programs, the defined outputs of the vertex shader code 502 and the defined inputs of the hull or geometry shader code 504 do not need to be explicitly "handled" by the APD 116. More specifically, because these defined inputs and outputs are hints to the APD 116 regarding the manner in which to transfer data between shader programs executing at different stages of the graphics processing pipeline 134, these defined inputs and outputs are not needed in the combined vertex and hull or geometry shader program 530. Thus, the driver 122 removes the defined outputs of the vertex shader code 502 and the defined inputs of the hull or geometry shader code 504 in creating the combined vertex and hull or geometry shader program 530. The driver 122 also modifies the instructions of the hull or geometry shader code 504 that read from inputs to instead read from locations in which outputs of the vertex shader code 502 are placed. For example, if certain registers would be indicated as storing outputs for the vertex shader code and certain registers would be indicated as storing inputs for the hull or geometry shader code, then the driver 122 modifies the instructions of the hull or geometry shader code to read from the registers indicated as storing outputs for the vertex shader code, instead of reading from registers indicated as storing inputs for the hull or geometry shader code.

[0052] In order to launch wavefronts to execute shader programs, the scheduler 136 reserves resources for those wavefronts. The resources include portions of various memory units, registers (such as vector registers 206 and scalar registers 208), entries in the wavefront bookkeeping 204 to keep track of the wavefronts, and other resources. The resources reserved for the combined vertex and hull or geometry shader program 530 differ from resources that would be reserved for independent vertex shader programs and hull or geometry shader programs in several ways. The number of wavefronts actually launched is different. The number of wavefronts to be launched for an independent vertex shader is dependent on the number of vertices 610 to be shaded. The number of

wavefronts to be launched for an independent hull or geometry shader is dependent on the number of patches (hull) or primitives (geometry) to be shaded. However, the number of wavefronts launched for the combined vertex and hull or geometry shader program 530 is less than the total number of wavefronts to launch for independent vertex shader programs and hull or geometry shader programs. This is because the combined shader program has instructions for both shader programs. Thus at least some of the combined shader programs that execute instructions for the vertex shader stage will also execute instructions for the hull or geometry shader stage.

[0053] In one example, the number of combined shader stage wavefronts to launch is based on the number of vertices to be processed in the vertex shader stage. More specifically, a number of work-items to launch is equal (or approximately equal) to the number of vertices 610 to process at the vertex shader stage. The number of wavefronts is based on the number of work-items to launch since each wavefront executes instructions for a fixed number of work-items. Additionally, because hull or geometry shader programs perform work on a collection of vertices 614, the number of work-items for processing vertices 610 in the vertex shader stage is greater than the number of work-items for performing hull or geometry shader operations. Further, the combined shader programs include instructions for both the vertex shader stage and the hull or geometry shader stage. Thus, at least some of the launched combined shader programs will execute instructions for both the vertex shader stage and the hull or geometry shader stage. For this reason, the number of combined shader wavefronts to launch is based on the number of vertices 610 to process in the vertex shader stage and is not based on the number of patches or primitives to process in the hull or geometry shader stage.

[0054] In addition, some of the registers or memory locations used for the vertex shader instructions are reused for the hull or geometry shader instructions. Thus, whereas independently executing wavefronts for vertex shader programs and hull or geometry shader programs would have to reserve registers and memory for the respective shader programs, a combined shader

program can reserve less memory and registers. In some examples, the amount of resources to reserve for the combined vertex and hull or geometry shader program 530 is, for each particular resource, the maximum of either the number of resources required by the vertex shader code 502 or the hull or geometry shader code 504. For example, if the vertex shader code needs 4 registers and the hull or geometry shader code 504 needs 8 registers, then the scheduler 136 reserves 8 registers. If the vertex shader code needs 100 bytes in the local data store memory 212 and the hull or geometry shader code needs 200 bytes in the local data store memory 212, then the scheduler 136 reserves 200 bytes. In addition, using the combined shader stages allows for a smaller amount of latency because stage-to-stage data is kept local to an execution unit. Thus, the latency associated with transmitting the data for a stage to an external memory unit for a first stage and then reading the data back in for a second stage is reduced.

[0055] Figure 6 illustrates operations for enabling or disabling wavefronts for the combined vertex and hull or geometry shader stage in order to accommodate that change in workload at the boundary between shader stages, according to an example. More specifically, Figure 6 illustrates a number of combined vertex and hull or geometry shader wavefronts 532 executing either a combined vertex and geometry shader program for the configuration of Figure 4A or a combined vertex and hull shader program for the configuration of Figure 4B (where this combined shader program is illustrated as V+H/G 530). The combined shader programs 530 include vertex shader portions 606 and hull or geometry shader portions 608. The vertex shader portions 606 accept vertices 510 as inputs and output shaded vertices 612 to the hull/geometry shader portions 608. The hull/geometry shader portions 608 accept the shaded vertices as input control points 614, and outputs output control points 616 in response. Note that because the shaded output vertices 612 are gathered together for processing to generate output patches, the number of wavefronts for the hull/geometry shader portion 608 is less than the number of vertices 614 for shading vertices by the vertex shader portion 606.

[0056] To accommodate this variation, a number of combined vertex and hull or geometry shader wavefronts 532 are launched in order to process the vertices identified for shading by the vertex shader stage. The combined shader wavefronts 532 are configured to have a changing execution footprint while executing. A "changing execution footprint" means that a different number of wavefronts 532 will execute for the vertex shader portoin 506 as compared with the hull or geometry shader portion 608. More specifically, vertex shader programs perform vertex shading operations for individual vertices. Although multiple instances of vertex shader programs corresponding to different work-items are executed in parallel on the programmable processing units 202, each individual work-item operates independently on only a single vertex of data, performing whatever transformations and other operations are specified for that vertex of data.

[0057] However, hull shader programs or geometry shader programs operate on multiple vertices of data (e.g., a collection of input control points for a hull shader or a collection of vertices for a geometry shader). More specifically, each work-item accepts multiple vertices as input control points and outputs a patch as multiple output control points. Because hull shader programs and geometry shader programs operate on multiple vertices, the overall number of work-items for performing hull shader and geometry shader operations is generally lower than the number of work-items for performing vertex shader operations.

[0058] Because of the difference in ratio between vertex shader work-items and hull shader or geometry shader work-items, many of the wavefronts 532 launched to execute the combined shader program are not in fact needed for the hull shader or geometry shader portion. Unneeded wavefronts are therefore put to sleep or terminated after executing the vertex shader portion 606. Putting the wavefronts 532 to sleep means continuing to track the wavefronts 532, storing state for the wavefronts 532 in memory, and the like, but not scheduling the sleeping wavefronts 532 for execution on any of the SIMD units 138. Terminating the wavefronts 532 means ending execution of the wavefronts 532

and discarding tracking data for the terminated wavefronts 532 that may be stored, for example, in wavefront bookkeeping 204.

[0059] In Figure 6, the left side of the figure illustrates the combined wavefronts 532 executing the vertex shader portion 606 and the right side of the figure illustrates the combined wavefronts 532 executing the hull or geometry shader portion 608 (inactive portions are illustrated with dotted lines). The right side of the figure illustrates the combined wavefronts 532 executing the hull or geometry shader program 508, with inactive wavefronts 532 and portions of shader programs indicated with dotted lines. Once the vertex shader portion 506 has been executed by the wavefronts 532, the APD 116 disables many of the wavefronts 532, which do not execute the hull or geometry shader portion 608. In the example of Figure 6, wavefront 532(1) executes the hull or geometry shader portion 608 and wavefronts 532(2) – 532(N) are disabled. In other examples, however, different numbers of wavefronts may be enabled and disabled for execution of the hull or geometry shader portion 608.

[0060] Each wavefront 532 that is not disabled gathers shaded vertices (as input control points 614) produced by multiple wavefronts 532 executing the vertex shader portion 606. This "gathering" can be accomplished by specific instructions inserted by the driver 122 into the combined shader program 530. More specifically, the driver 122 can insert an instruction that reads from the locations of the modified vertices as specified for multiple wavefronts 532 in the vertex shader portion 606.

[0061] To ensure that the hull or geometry shader portion 608 of the combined shader program has all of the input control points 614 from the vertex shader portions 606, the driver 122 inserts a barrier instruction between the vertex shader portion 606 and the hull or geometry shader portion 608 when generating the combined vertex and hull or geometry shader program 530. The barrier instruction causes processing in the hull or geometry shader portion 608 to wait to execute until the wavefronts producing the data for the hull or geometry shader portion 608 have finished executing the vertex shader portion 606.

[0062] Figure 6 shows transition from vertex to hull or geometry portions of the combined shader program 530 and the corresponding disabling of wavefronts. It should be understood, however, that the combined vertex and hull or geometry shader programs 530 may include many transitions that cause various wavefronts to sleep and wake up.

[0063] In another example, variation in workload within the combined vertex and hull or geometry shader program 530 is accommodated by enabling or disabling work-items within the wavefronts. More specifically, each wavefront performs work on a set of vertices, and then the same wavefront performs work on a set of primitives. Because each work-item works on a single vertex for vertex operations and on a single primitive for primitive operations, generally speaking, the number of work-items for primitive processing is less than the number of work-items for vertex processing. To accommodate this difference, the wavefront disables unneeded work-items, with the remaining work-items performing the primitive operations.

[0064] Figure 7 is a flow diagram of a method 700 for executing a combined vertex and hull or geometry shader program for a combined vertex and hull or geometry shader stage, according to an example. Although described with respect to the system shown and described with respect to Figures 1-6, it should be understood that any system configured to perform the method, in any technically feasible order, falls within the scope of the present disclosure.

[0065] As shown, the method 700 begins at step 702, where the APD 116 obtains a combined shader program for a combined vertex and hull or geometry shader stage. The combined shader program may be generated by a driver 122 based on vertex shader code and hull shader code or geometry shader code received from, e.g., an application 126. More specifically, the driver 122 compiles the vertex shader code and hull shader code or geometry shader code to generate vertex and hull or geometry shader compiled instructions and combines the vertex and hull or geometry shader instructions into a single combined shader program. The driver 122 may modify aspects of the individual shader programs,

such as the specified outputs for the vertex shader code and the specified inputs for the hull shader code or geometry shader code.

[0066] At step 704, the APD 116 reserves resources for execution of the combined shader program. Reserved resources include memory, such as registers and memory in a local data store memory 212, entries in wavefront bookkeeping 204 for tracking executing wavefronts, and other resources. The number of resources reserved for the combined shader program may be less than the number of resources reserved for individual vertex and hull or geometry shader programs because some such resources (e.g., registers, memory) can be reused from the vertex shader portion of the combined shader program to the hull or geometry shader portion of the combined shader program.

[0067] At step 706, the APD 116 spawns and launches wavefronts to execute the combined shader program. Each wavefront executes the combined shader program, which includes instructions that are based on both the vertex shader code and the hull or geometry shader code. The number of wavefronts launched is based on the greater of the number of wavefronts to execute the vertex shader code and the number of wavefronts to execute the hull or geometry shader code. Typically, more wavefronts are used to execute vertex shader code, so the number of wavefronts spawned is dependent on the number of vertices to process in the vertex shader portion of the combined shader program.

[0068] A method of executing a shader program for a combined shader stage of a graphics processing pipeline is provided. The method includes retrieving a combined shader program for the combined shader stage, reserving resources for a plurality of wavefronts to execute the combined shader program, and spawning the plurality of wavefronts to execute the combined shader program. The combined shader stage includes one of a combined vertex shader stage and hull shader stage and a combined vertex shader stage and geometry shader stage.

[0069] An accelerated processing device (APD) for executing a shader program for a combined shader stage of a graphics processing pipeline is provided. The APD includes a plurality of shader engines including registers

and local data store memory and a scheduler. The scheduler is configured to retrieve a combined shader program for the combined shader stage, reserve resources for a plurality of wavefronts to execute the combined shader program in the plurality of shader engines, and spawning the plurality of wavefronts to execute the combined shader program in the plurality of shader engines. The combined shader stage includes one of a combined vertex shader stage and hull shader stage and a combined vertex shader stage and geometry shader stage.

[0070] A computer system for executing a shader program for a combined shader stage of a graphics processing pipeline is provided. The computer system includes a processor executing a device driver for controlling an accelerated processing device (APD) and the APD. The APD includes a plurality of shader engines including registers and local data store memory and a scheduler. The scheduler is configured to receive a combined shader program for the combined shader stage from the device driver, reserve resources for a plurality of wavefronts to execute the combined shader program in the plurality of shader engines, and spawn the plurality of wavefronts to execute the combined shader program in the plurality of shader engines. The combined shader stage includes one of a combined vertex shader stage and hull shader stage and a combined vertex shader stage and geometry shader stage.

[0071] It should be understood that many variations are possible based on the disclosure herein. Although features and elements are described above in particular combinations, each feature or element may be used alone without the other features and elements or in various combinations with or without other features and elements.

[0072] The methods provided may be implemented in a general purpose computer, a processor, or a processor core. Suitable processors include, by way of example, a general purpose processor, a special purpose processor, a conventional processor, a digital signal processor (DSP), a plurality of microprocessors, one or more microprocessors in association with a DSP core, a controller, a microcontroller, Application Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGAs) circuits, any other type of integrated

circuit (IC), and/or a state machine. Such processors may be manufactured by configuring a manufacturing process using the results of processed hardware description language (HDL) instructions and other intermediary data including netlists (such instructions capable of being stored on a computer readable media). The results of such processing may be maskworks that are then used in a semiconductor manufacturing process to manufacture a processor which implements aspects of the embodiments.

[0073] The methods or flow charts provided herein may be implemented in a computer program, software, or firmware incorporated in a non-transitory computer-readable storage medium for execution by a general purpose computer or a processor. Examples of non-transitory computer-readable storage mediums include a read only memory (ROM), a random access memory (RAM), a register, cache memory, semiconductor memory devices, magnetic media such as internal hard disks and removable disks, magneto-optical media, and optical media such as CD-ROM disks, and digital versatile disks (DVDs).

* * *

CLAIMS

What is claimed is:

1. A method of executing a shader program for a combined shader stage of a graphics processing pipeline, the method comprising:

retrieving a combined shader program for the combined shader stage;

reserving resources for a plurality of wavefronts to execute the combined shader program; and

spawning the plurality of wavefronts to execute the combined shader program,

wherein the combined shader stage includes one of a combined vertex shader stage and hull shader stage and a combined vertex shader stage and geometry shader stage.

2. The method of claim 1, wherein:

the combined shader stage is part of a graphics processing pipeline in which tessellation is disabled; and

the combined shader stage comprises the combined vertex shader stage and geometry shader stage.

3. The method of claim 1, wherein:

the combined shader stage is part of a graphics processing pipeline in which tessellation is enabled; and

the combined shader stage comprises the combined vertex shader stage and hull shader stage.

4. The method of claim 1, wherein the combined shader program comprises a single shader program that includes instructions from both a vertex shader program and either a hull shader program or a geometry shader program.

5. The method of claim 4, wherein:

the vertex shader program specifies vertex shader outputs that indicate to graphics processing hardware locations of vertex shader output values to transfer to a subsequent stage in the graphics processing pipeline; and

the combined shader program does not include the vertex shader outputs.

6. The method of claim 4, wherein reserving the resources comprises:

selecting the greater number of registers of a first number of registers for executing the vertex shader program and a second number of registers for executing either the hull shader program or the geometry shader program; and

reserving the greater number of registers.

7. The method of claim 4, further comprising:

receiving an indication of a number of vertices to be processed by a vertex shader portion of the combined shader stage and either a number of patches to be processed by the hull shader stage or a number of primitives to be processed by the geometry shader stage; and

determining a number of wavefronts for processing the number of vertices by the vertex shader portion of the combined shader stage,

wherein spawning the plurality of wavefronts to execute the combined shader program comprises spawning the determined number of wavefronts to execute the combined shader program.

8. The method of claim 7, further comprising:

disabling one or more of the wavefronts based on a ratio of a first number of work-items to execute the vertex shader program and a second number of work-items to execute either the hull shader program or the geometry shader program,

wherein disabling comprises one of causing the one or more of the wavefronts to sleep or terminating the one or more wavefronts.

9. The method of claim 1, wherein:

the combined shader program includes a barrier instruction between a vertex shader portion and either a hull shader portion or a geometry shader portion, and

the barrier instruction causes processing in the vertex shader portion to wait to begin until processing in the hull shader portion or geometry shader portion is complete.

10. An accelerated processing device (APD) for executing a shader program for a combined shader stage of a graphics processing pipeline, the APD comprising:

a plurality of shader engines including registers and local data store memory; and

a scheduler configured to:

retrieve a combined shader program for the combined shader stage;

reserve resources for a plurality of wavefronts to execute the combined shader program in the plurality of shader engines; and

spawning the plurality of wavefronts to execute the combined shader program in the plurality of shader engines,

wherein the combined shader stage includes one of a combined vertex shader stage and hull shader stage and a combined vertex shader stage and geometry shader stage.

11. The APD of claim 10, wherein:

the combined shader stage is part of a graphics processing pipeline in which tessellation is disabled; and

the combined shader stage comprises the combined vertex shader stage and geometry shader stage.

12. The APD of claim 10, wherein:

the combined shader stage is part of a graphics processing pipeline in which tessellation is enabled; and

the combined shader stage comprises the combined vertex shader stage and hull shader stage.

13. The APD of claim 10, wherein the combined shader program comprises a single shader program that includes instructions from both a vertex shader program and either a hull shader program or a geometry shader program.

14. The APD of claim 13, wherein:

the vertex shader program specifies vertex shader outputs that indicate to graphics processing hardware locations of vertex shader output values to transfer to a subsequent stage in the graphics processing pipeline; and

the combined shader program does not include the vertex shader outputs.

15. The APD of claim 13, wherein the scheduler is configured to reserve the resources by:

selecting the greater number of registers of a first number of registers for executing the vertex shader program and a second number of registers for executing either the hull shader program or the geometry shader program; and

reserving the greater number of registers.

16. The APD of claim 13, wherein the scheduler is further configured to:

receive an indication of a number of vertices to be processed by a vertex shader portion of the combined shader stage and either a number of patches to be processed by the hull shader stage or a number of primitives to be processed by the geometry shader stage; and

determine a number of wavefronts for processing the number of vertices by the vertex shader portion of the combined shader stage,

wherein spawning the plurality of wavefronts to execute the combined shader program comprises spawning the determined number of wavefronts to execute the combined shader program.

17. The APD of claim 16, wherein the scheduler is further configured to:

disable one or more of the wavefronts based on a ratio of a first number of work-items to execute the vertex shader program and a second number of work-items to execute either the hull shader program or the geometry shader program,

wherein disabling comprises one of causing the one or more of the wavefronts to sleep or terminating the one or more wavefronts.

18. The APD of claim 10, wherein:

the combined shader program includes a barrier instruction between a vertex shader portion and either a hull shader portion or a geometry shader portion, and

the barrier instruction causes processing in the vertex shader portion to wait to begin until processing in the hull shader portion or geometry shader portion is complete.

19. A computer system for executing a shader program for a combined shader stage of a graphics processing pipeline, the computer system comprising:

a processor executing a device driver for controlling an accelerated processing device (APD); and

the APD, comprising:

a plurality of shader engines including registers and local data store memory; and

a scheduler configured to:

receive a combined shader program for the combined shader stage from the device driver;

reserve resources for a plurality of wavefronts to execute the combined shader program in the plurality of shader engines; and

spawn the plurality of wavefronts to execute the combined shader program in the plurality of shader engines,

wherein the combined shader stage includes one of a combined vertex shader stage and hull shader stage and a combined vertex shader stage and geometry shader stage.

20. The computing device of claim 19, wherein:

the combined shader stage is part of a graphics processing pipeline in which tessellation is disabled; and

the combined shader stage comprises the combined vertex shader stage and geometry shader stage.

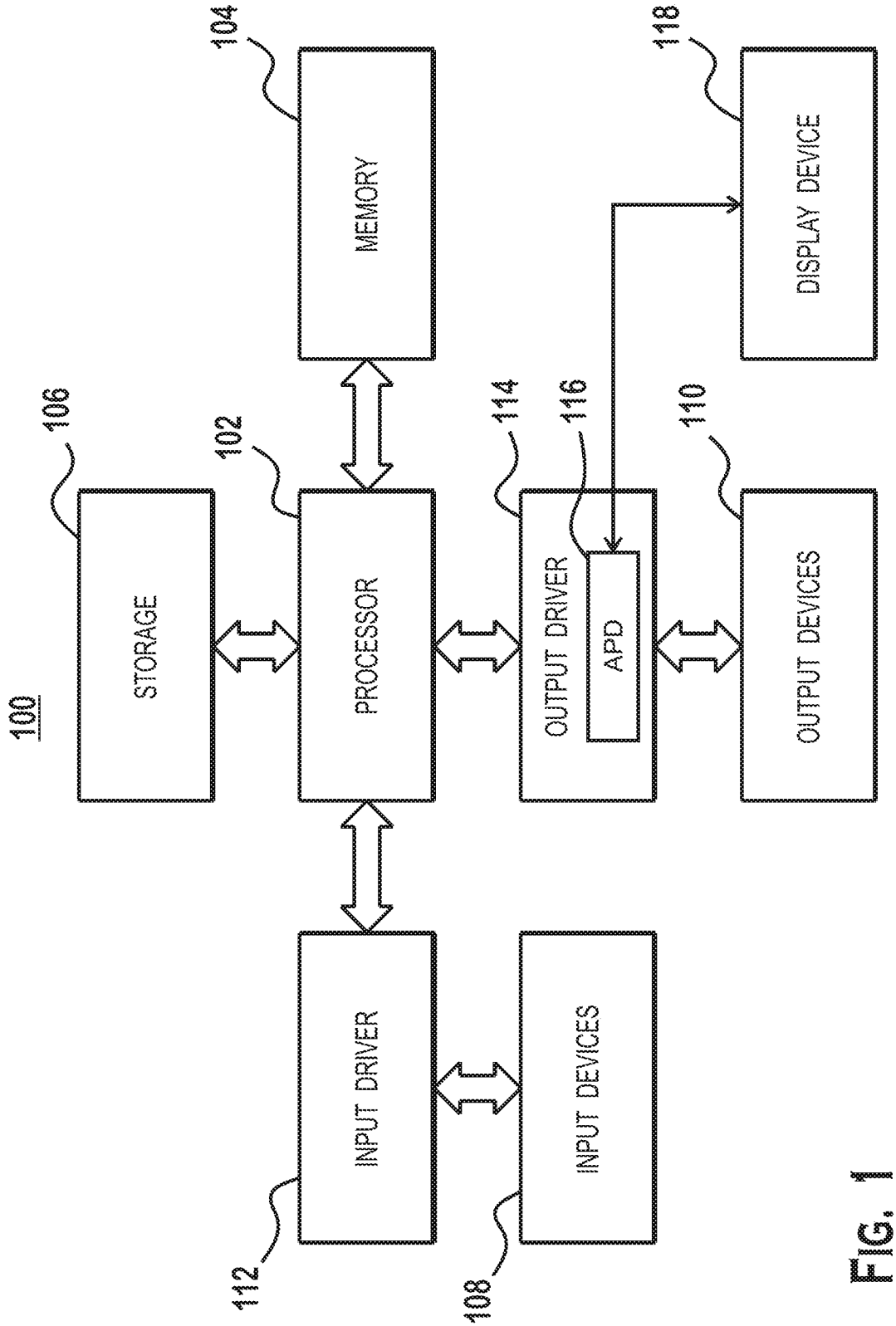

100
STORAGE
106
PROCESSOR
102
MEMORY
104
OUTPUT DRIVER
114
APD
116
OUTPUT DEVICES
110
DISPLAY DEVICE
118
INPUT DRIVER
112
INPUT DEVICES
108

FIG. 1

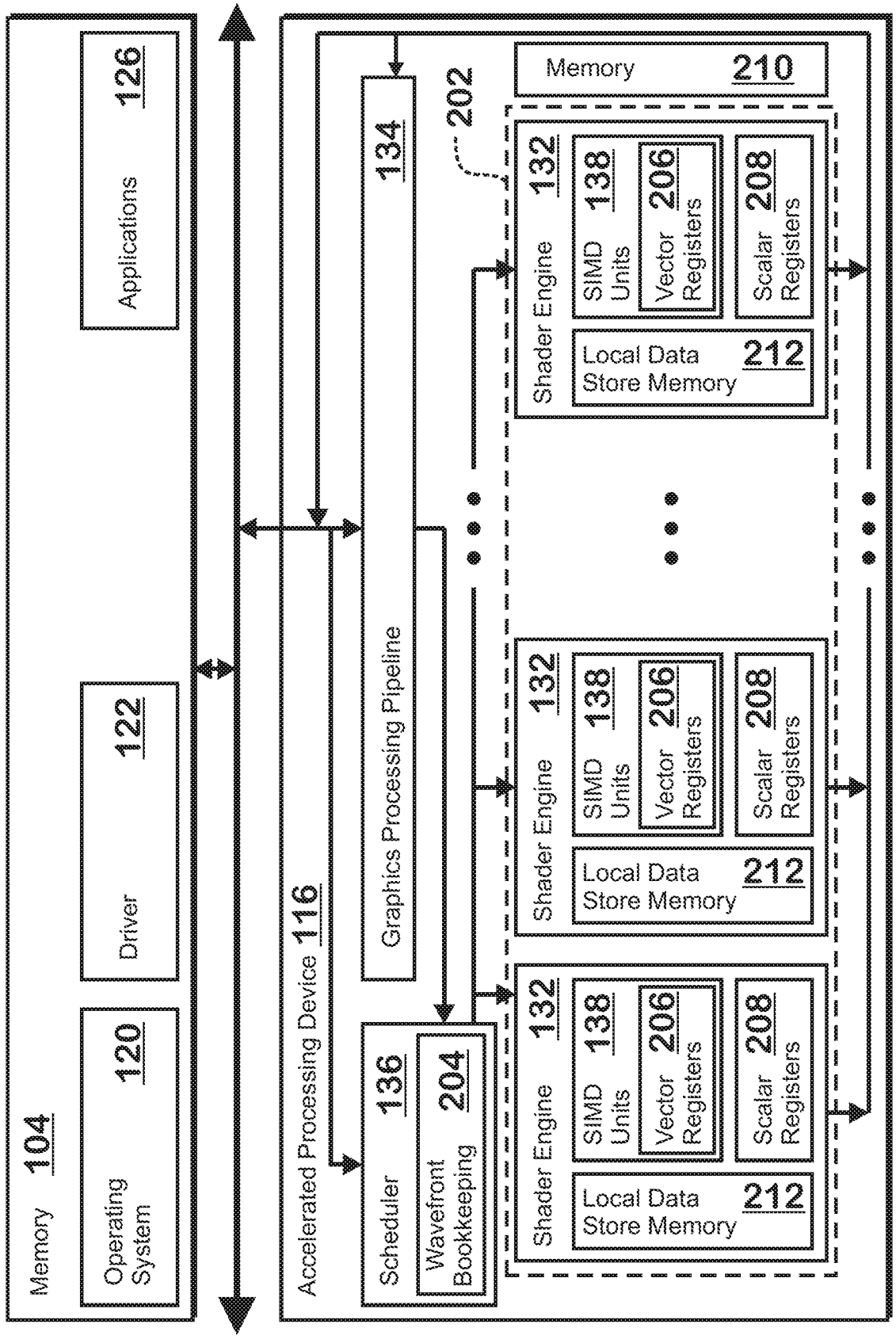

Memory 104
Operating System 120
Driver 122
Applications 126
Accelerated Processing Device 116
Graphics Processing Pipeline 134
Scheduler 136
Wavefront Bookkeeping 204
202
Memory 210
Shader Engine 132
SIMD Units 138
Vector Registers 206
Scalar Registers 208
Local Data Store Memory 212

FIG. 2

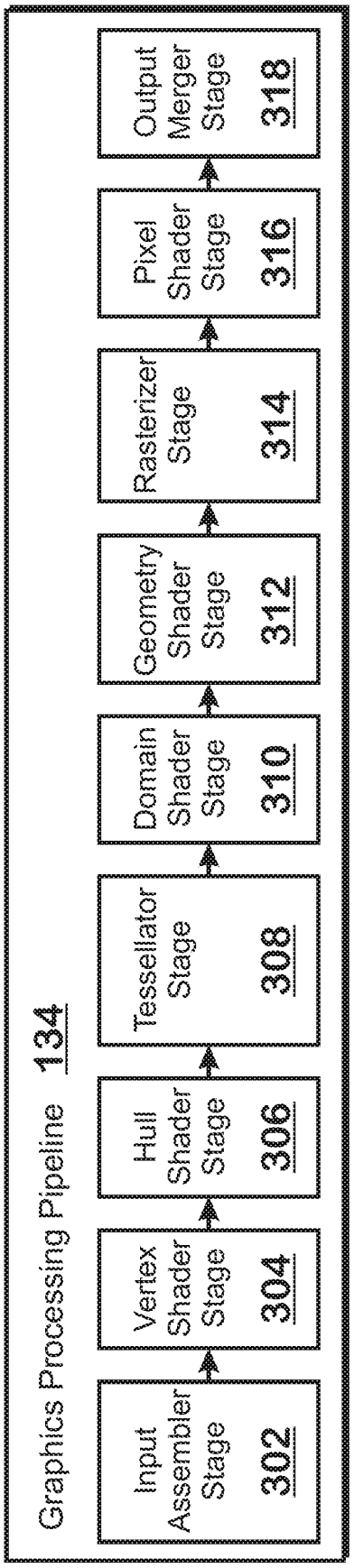

Graphics Processing Pipeline 134
Input Assembler Stage 302
Vertex Shader Stage 304
Hull Shader Stage 306
Tessellator Stage 308
Domain Shader Stage 310
Geometry Shader Stage 312
Rasterizer Stage 314
Pixel Shader Stage 316
Output Merger Stage 318

FIG. 3

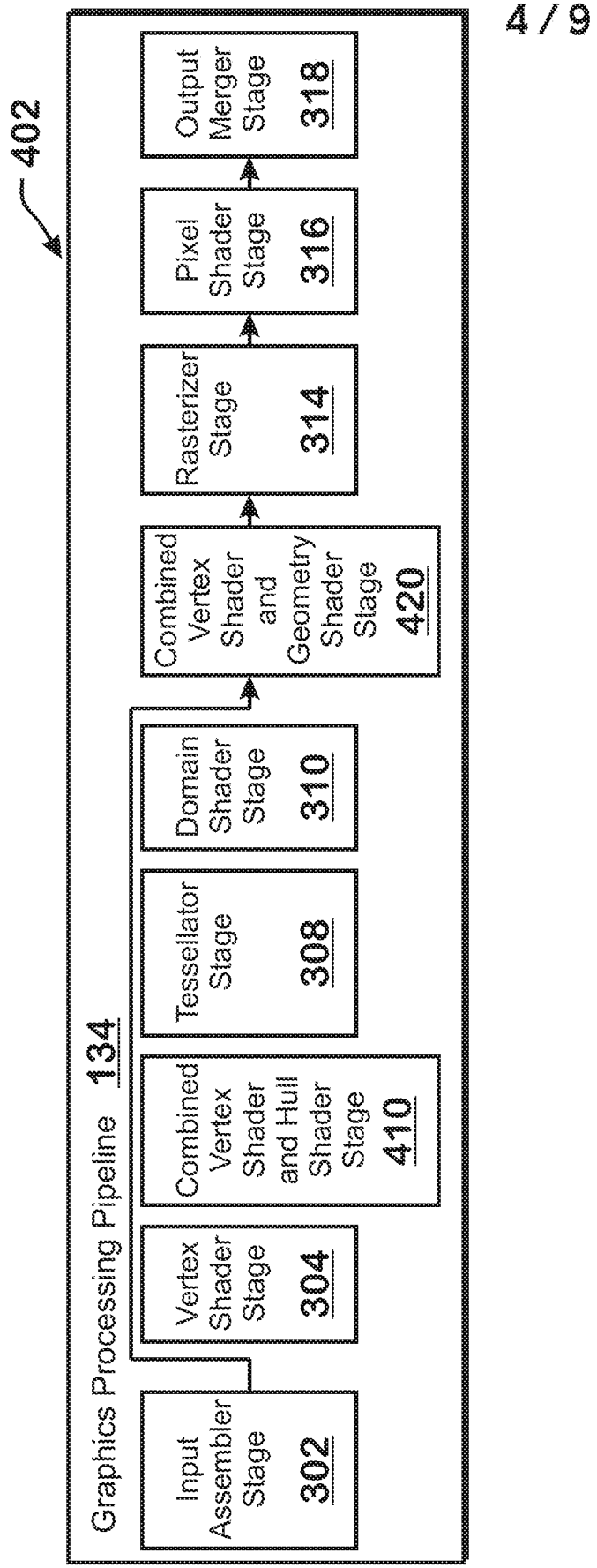

402
Graphics Processing Pipeline 134
Input Assembler Stage 302
Vertex Shader Stage 304
Combined Vertex Shader and Hull Shader Stage 410
Tessellator Stage 308
Domain Shader Stage 310
Combined Vertex Shader and Geometry Shader Stage 420
Rasterizer Stage 314
Pixel Shader Stage 316
Output Merger Stage 318

FIG. 4A

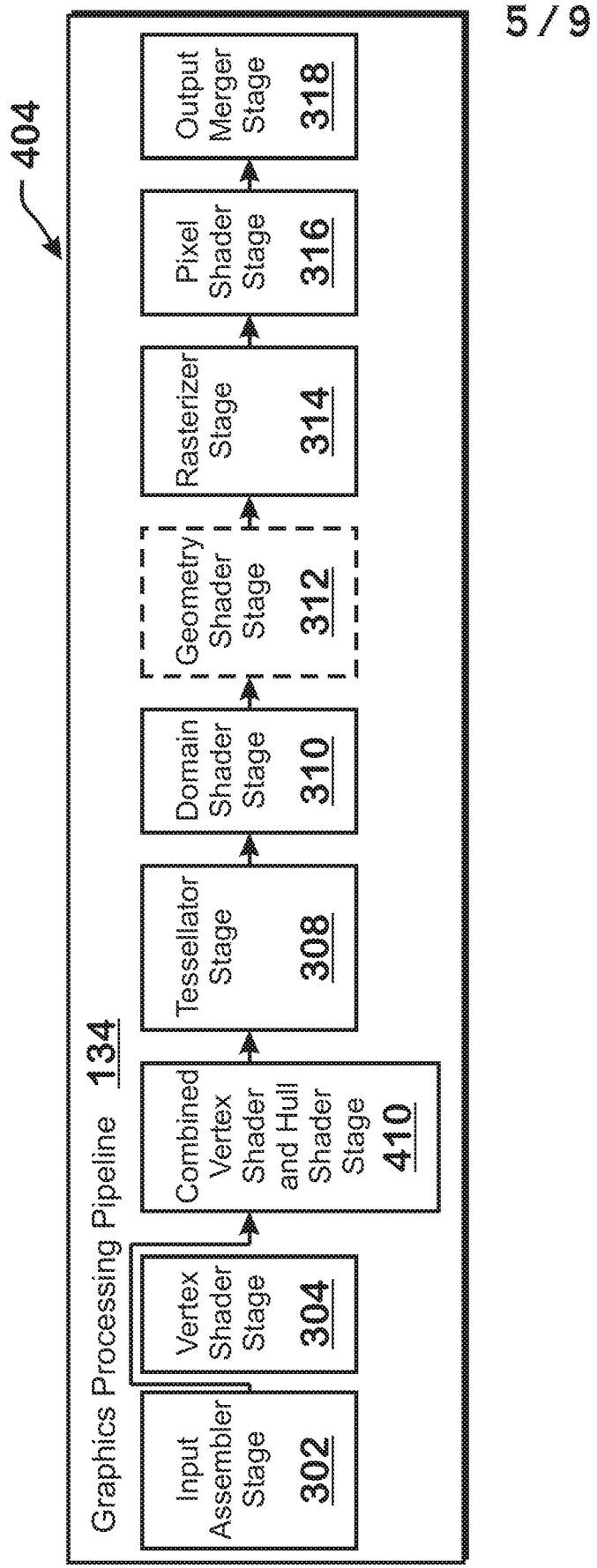

404
Graphics Processing Pipeline 134
Input Assembler Stage 302
Vertex Shader Stage 304
Combined Vertex Shader and Hull Shader Stage 410
Tessellator Stage 308
Domain Shader Stage 310
Geometry Shader Stage 312
Rasterizer Stage 314
Pixel Shader Stage 316
Output Merger Stage 318

FIG. 4B

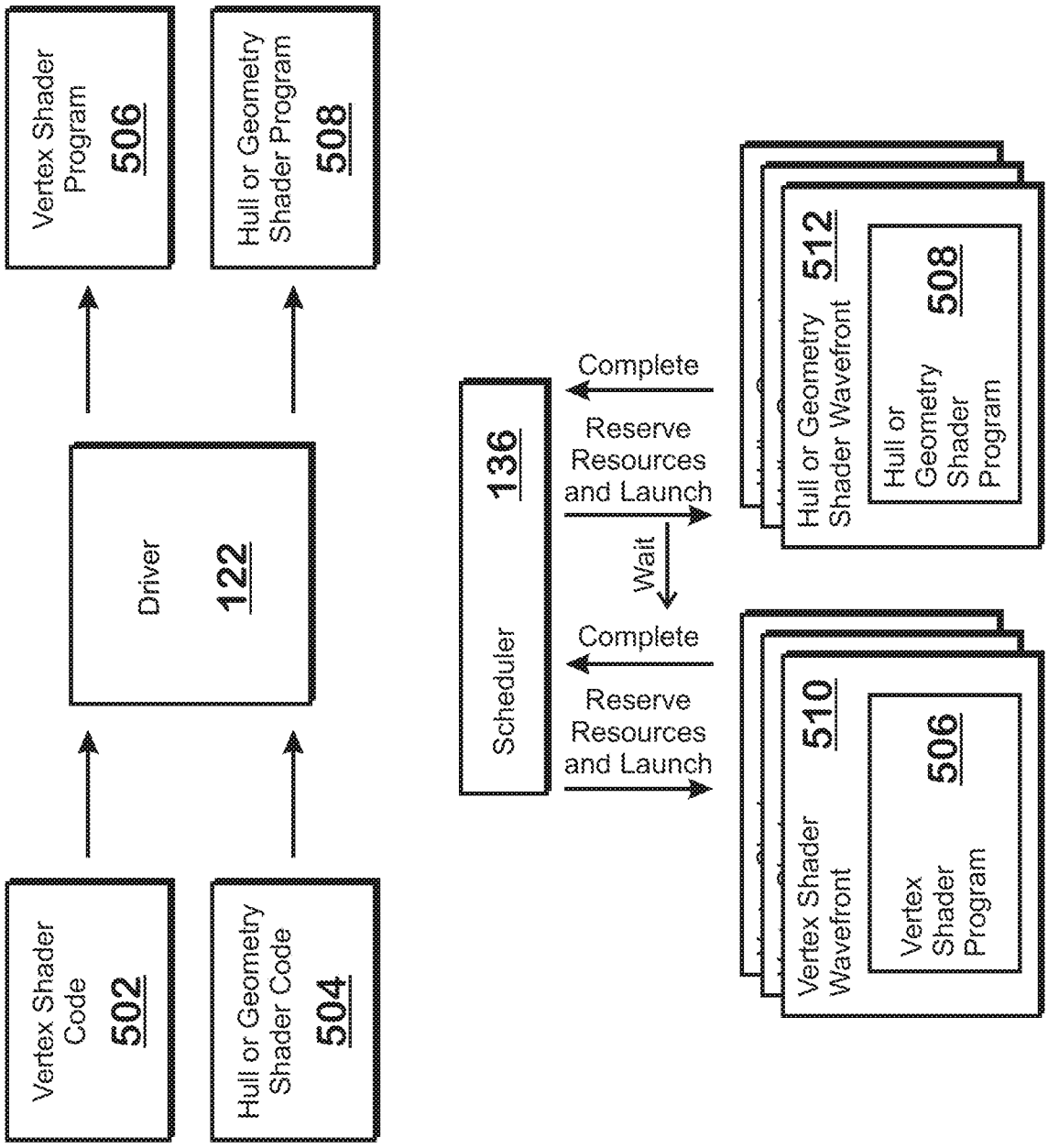

Vertex Shader Code 502
Hull or Geometry Shader Code 504
Driver 122
Vertex Shader Program 506
Hull or Geometry Shader Program 508
Scheduler 136
Reserve Resources and Launch
Complete
Wait
Reserve Resources and Launch
Complete
Vertex Shader Wavefront 510
Vertex Shader Program 506
Hull or Geometry Shader Wavefront 512
Hull or Geometry Shader Program 508

FIG. 5A

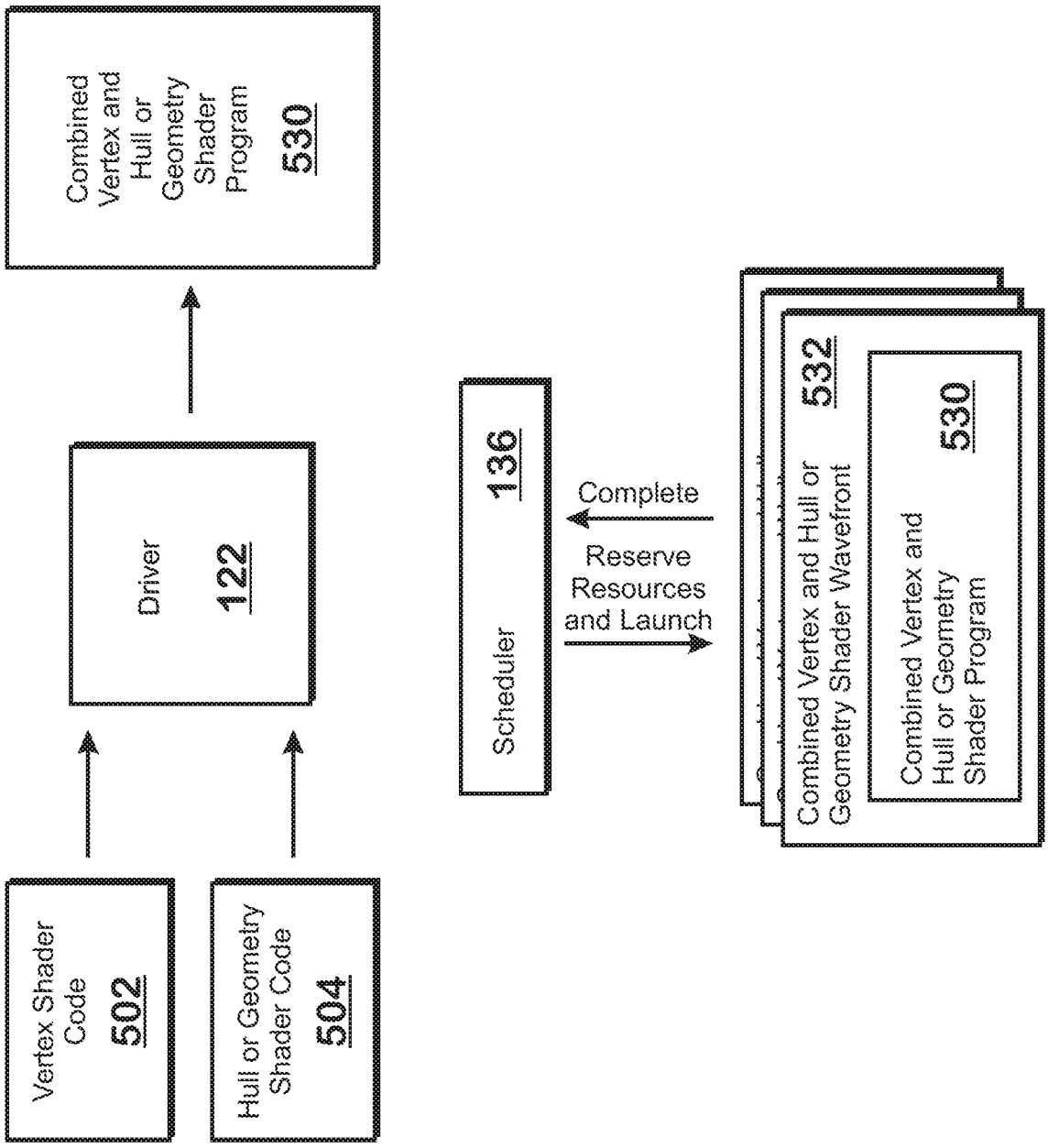

Vertex Shader Code 502
Hull or Geometry Shader Code 504
Driver 122
Combined Vertex and Hull or Geometry Shader Program 530
Scheduler 136
Reserve Resources and Launch
Complete
Combined Vertex and Hull or Geometry Shader Wavefront 532
Combined Vertex and Hull or Geometry Shader Program 530

FIG. 5B

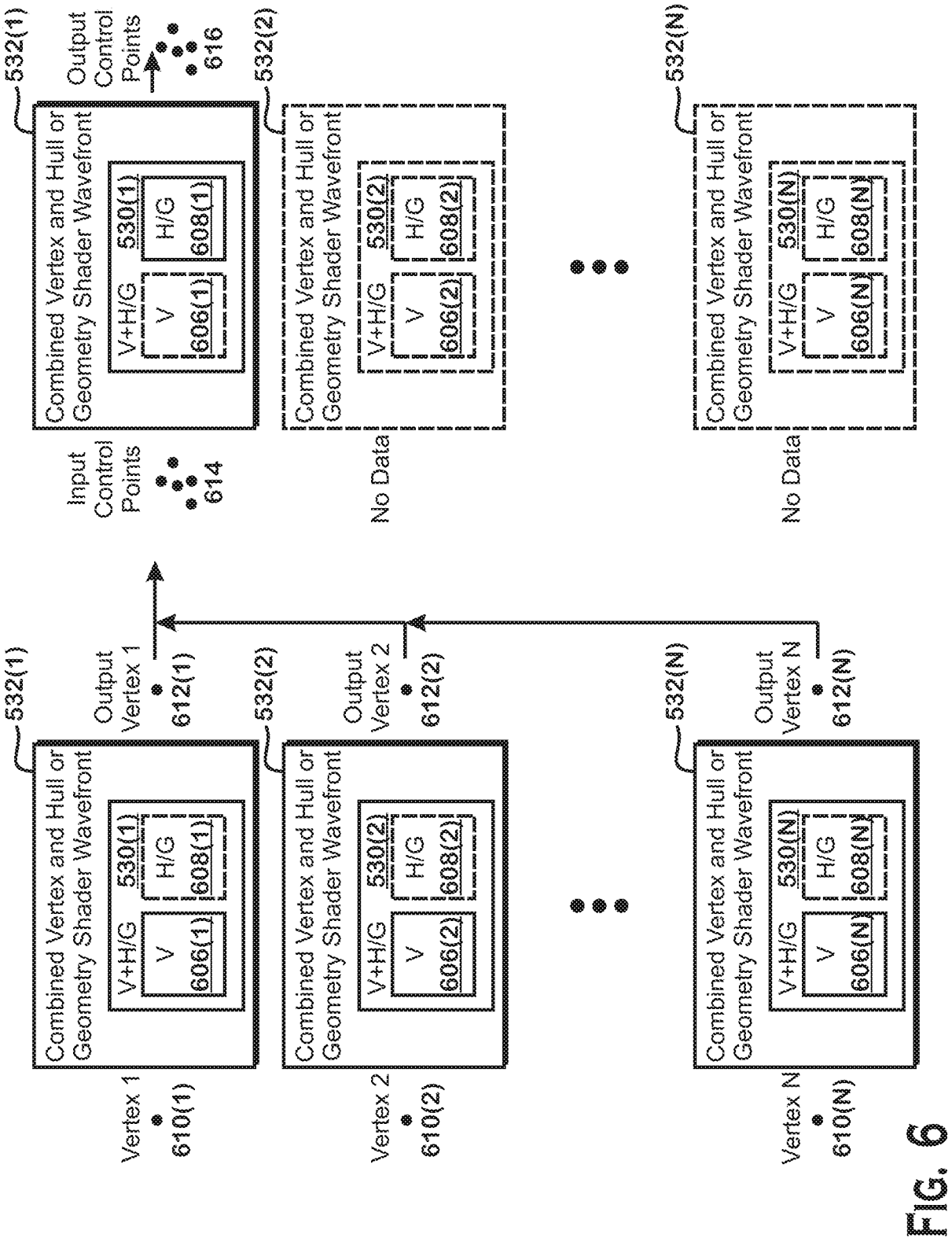

532(1)
Combined Vertex and Hull or Geometry Shader Wavefront
V+H/G
530(1)
V
606(1)
H/G
608(1)
Output Control Points
616
Input Control Points
614
Output Vertex 1
612(1)
Vertex 1
610(1)
532(2)
Combined Vertex and Hull or Geometry Shader Wavefront
V+H/G
530(2)
V
606(2)
H/G
608(2)
No Data
Output Vertex 2
612(2)
Vertex 2
610(2)
532(N)
Combined Vertex and Hull or Geometry Shader Wavefront
V+H/G
530(N)
V
606(N)
H/G
608(N)
No Data
Output Vertex N
612(N)
Vertex N
610(N)

FIG. 6

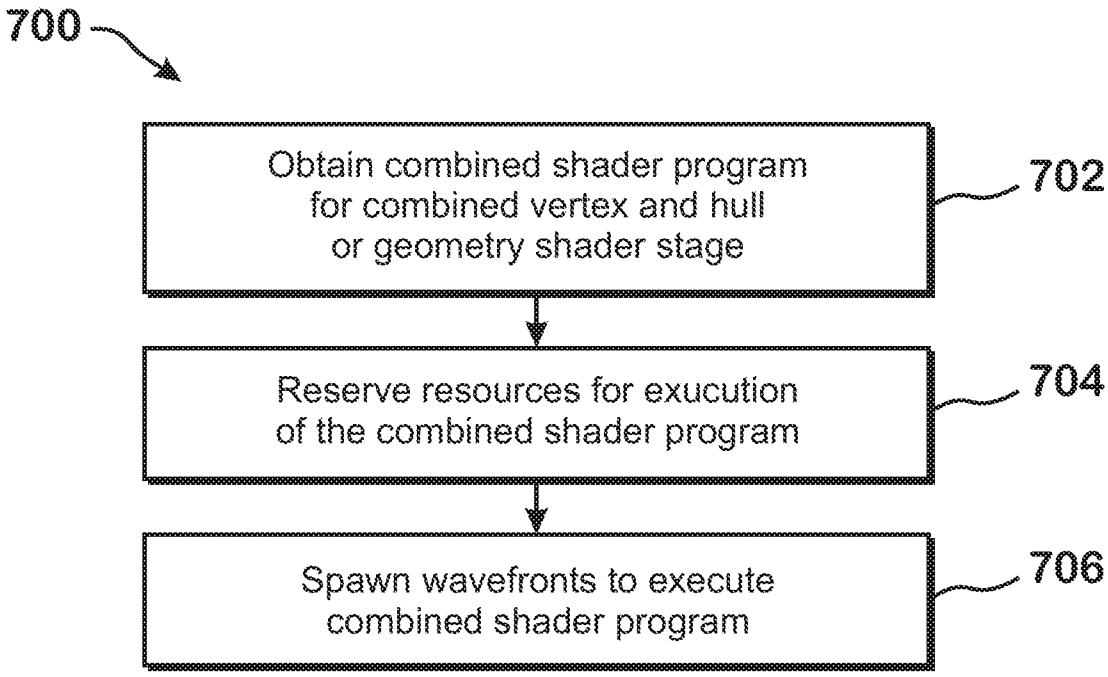

700
Obtain combined shader program for combined vertex and hull or geometry shader stage
702
Reserve resources for exucution of the combined shader program
704
Spawn wavefronts to execute combined shader program
706

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2017/050345

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06T15/00
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06T

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	Anonymous: "AMD GRAPHICS CORES NEXT (GCN) ARCHITECTURE", White Paper - AMD, 30 June 2012 (2012-06-30), XP055217702, Retrieved from the Internet: URL:https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf [retrieved on 2015-10-01] pages 3,12	1-20
Y	US 2013/265307 A1 (GOEL VINEET [US] ET AL) 10 October 2013 (2013-10-10) paragraph [0079]	1-20
Y	US 2008/094408 A1 (YIN XIAOQIN [TW] ET AL) 24 April 2008 (2008-04-24) paragraph [0022]	1-20

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
5 October 2017	13/10/2017

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	dos Santos, Luís

1

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	Matthias Trapp and Jürgen Döllner: "Automated Combination of Real-Time Shader Programs", , 2007, XP002771208, Retrieved from the Internet: URL:https://pdfs.semanticscholar.org/82ae/a3d4b29012dc229122915aa2367994b1dc0d.pdf [retrieved on 2017-06-20] abstract -----	1-20

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2017/050345

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013265307 A1	10-10-2013	CN 104205174 A	10-12-2014
		CN 104246829 A	24-12-2014
		CN 104813367 A	29-07-2015
		EP 2834792 A2	11-02-2015
		EP 2834793 A2	11-02-2015
		EP 2834794 A2	11-02-2015
		JP 6162215 B2	12-07-2017
		JP 6162216 B2	12-07-2017
		JP 2015524092 A	20-08-2015
		JP 2015529859 A	08-10-2015
		JP 2015529860 A	08-10-2015
		KR 20150002738 A	07-01-2015
		KR 20150002742 A	07-01-2015
		KR 20150002745 A	07-01-2015
		US 2013265307 A1	10-10-2013
		US 2013265308 A1	10-10-2013
		US 2013265309 A1	10-10-2013
		WO 2013151748 A2	10-10-2013
		WO 2013151750 A2	10-10-2013
		WO 2013151751 A2	10-10-2013
US 2008094408 A1	24-04-2008	CN 101017566 A	15-08-2007
		US 2008094408 A1	24-04-2008