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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 December 2021 (30.12.2021)

WIPO | PCT

(10) International Publication Number
WO 2021/262434 A1

(51) International Patent Classification:
G06T 15/00 (2006.01) *G06T 7/50* (2017.01)
G06T 15/04 (2011.01) *G06T 7/90* (2017.01)
G06T 15/40 (2011.01)

(21) International Application Number:
PCT/US2021/036485

(22) International Filing Date:
08 June 2021 (08.06.2021)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
63/044,703 26 June 2020 (26.06.2020) US
17/028,811 22 September 2020 (22.09.2020) US

(71) Applicant: **ADVANCED MICRO DEVICES, INC.** [US/US]; 2485 Augustine Drive, Santa Clara, California 95054 (US).

(72) Inventors: **BRENNAN, Christopher J.**; c/o Advanced Micro Devices, Inc., 90 Central St., Floors 1, 2 & 3, Boxborough, Massachusetts 01719 (US). **GHODRAT, Fataneh F.**; c/o Advanced Micro Devices, Inc., 90 Central St., Floors 1, 2 & 3, Boxborough, Massachusetts 01719 (US). **WEI, Tien En**; c/o Advanced Micro Devices, Inc., 90 Central St., Floors 1, 2 & 3, Boxborough, Massachusetts 01719 (US).

(74) Agent: **GUSHUE, Joseph P.**; Volpe Koenig, 30 South 17th Street, Duane Morris 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, IT, 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, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, 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,

(54) Title: LOAD INSTRUCTION FOR MULTI SAMPLE ANTI-ALIASING

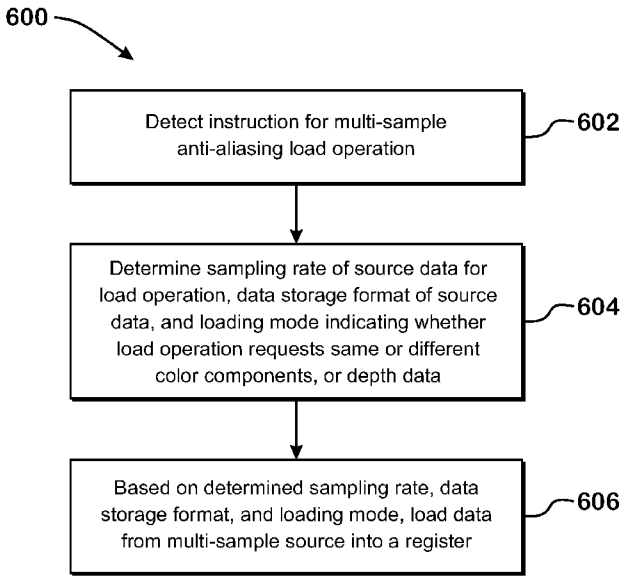

FIG. 6

(57) Abstract: Techniques for performing multi-sample anti-aliasing operations are provided. The techniques include detecting an instruction for a multi-sample anti-aliasing load operation; determining a sampling rate of source data for the load operation, data storage format of the source data, and loading mode indicating whether the load operation requests same or different color components, or depth data; and based on the determined sampling rate, data storage format, and loading mode, load data from a multi-sample source into a register.

WO 2021/262434 A1

[Continued on next page]

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))*

LOAD INSTRUCTION FOR MULTI SAMPLE ANTI-ALIASING

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional application No. 63/044,703, entitled "LOAD INSTRUCTION FOR MULTI SAMPLE ANTI-ALIASING," filed on June 26, 2020 and U.S. Non-Provisional Application No. 17/028,811, entitled "LOAD INSTRUCTION FOR MULTI SAMPLE ANTI-ALIASING," filed on September 22, 2020, the entirety of which are hereby incorporated herein by reference.

BACKGROUND

[0002] Three-dimensional ("3D") graphics processing pipelines perform a series of steps to convert input geometry into a two-dimensional ("2D") image for display on a screen. In multi-sample anti-aliasing, a high resolution image is generated and then "resolved" into a lower resolution image. Improvements to this technology are constantly being made.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] A more detailed understanding can be had from the following description, given by way of example in conjunction with the accompanying drawings wherein:

[0004] Figure 1 is a block diagram of an example device in which one or more features of the disclosure can be implemented;

[0005] Figure 2 illustrates details of the device of Figure 1, according to an example;

[0006] Figure 3 is a block diagram showing additional details of the graphics processing pipeline illustrated in Figure 2;

[0007] Figure 4 illustrates a multi sample anti-aliasing load operation 400, according to an example;

[0008] Figures 5A-5C illustrate example variations for the multi-sample load instruction;

[0009] Figure 5D illustrates a different data layout than that shown in Figures 5A-5C; and

[0010] Figure 6 is a flow diagram of a method 600 for performing multi-sample anti-aliasing operations, according to an example.

DETAILED DESCRIPTION

[0011] Techniques for performing multi-sample anti-aliasing operations are provided. The techniques include detecting an instruction for a multi-sample anti-aliasing load operation; determining a sampling rate of source data for the load operation, data storage format of the source data, and loading mode indicating whether the load operation requests same or different color components, or depth data; and based on the determined sampling rate, data storage format, and loading mode, load data from a multi-sample source into a register.

[0012] Figure 1 is a block diagram of an example device 100 in which one or more features of the disclosure can be implemented. The device 100 could be one of, but is not limited to, for example, a computer, a gaming device, a handheld device, a set-top box, a television, a mobile phone, a tablet computer, or other computing device. The device 100 includes a processor 102, a memory 104, a storage 106, one or more input devices 108, and one or more output devices 110. The device 100 also includes one or more input drivers 112 and one or more output drivers 114. Any of the input drivers 112 are embodied as hardware, a combination of hardware and software, or software, and serve the purpose of controlling input devices 112 (e.g., controlling operation, receiving inputs from, and providing data to input drivers 112). Similarly, any of the output drivers 114 are embodied as hardware, a combination of hardware and software, or software, and serve the purpose of controlling output devices 114 (e.g., controlling operation, receiving inputs from, and providing data to output drivers 114). It is understood that the device 100 can include additional components not shown in Figure 1.

[0013] In various alternatives, 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 can

be a CPU or a GPU. In various alternatives, the memory 104 is located on the same die as the processor 102, or is 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.

[0014] The storage 106 includes a fixed or removable storage, for example, without limitation, a hard disk drive, a solid state drive, an optical disk, or a flash drive. The input devices 108 include, without limitation, 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, without limitation, 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).

[0015] The input driver 112 and output driver 114 include one or more hardware, software, and/or firmware components that are configured to interface with and drive input devices 108 and output devices 110, respectively. 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, which, in some examples, is a physical display device or a simulated device that uses a remote display protocol to show output. The APD 116 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. As described in further detail below, the APD 116 includes one or more parallel processing units configured to perform computations in accordance with a single-instruction-multiple-data ("SIMD") paradigm. Thus, although various functionality is described herein as being performed by or in conjunction with the APD 116, in

various alternatives, the functionality described as being performed by the APD 116 is additionally or alternatively performed by other computing devices having similar capabilities that are not driven by a host processor (e.g., processor 102) and configured to provide graphical output to a display device 118. For example, it is contemplated that any processing system that performs processing tasks in accordance with a SIMD paradigm may be configured to perform the functionality described herein. Alternatively, it is contemplated that computing systems that do not perform processing tasks in accordance with a SIMD paradigm performs the functionality described herein.

[0016] Figure 2 illustrates details of the device 100 and the APD 116, according to an example. The processor 102 (Figure 1) executes an operating system 120, a driver 122, and applications 126, and may also execute other software alternatively or additionally. The operating system 120 controls various aspects of the device 100, such as managing hardware resources, processing service requests, scheduling and controlling process execution, and performing other operations. The APD driver 122 controls operation of the APD 116, sending tasks such as graphics rendering tasks or other work to the APD 116 for processing. The APD driver 122 also includes a just-in-time compiler that compiles programs for execution by processing components (such as the SIMD units 138 discussed in further detail below) of the APD 116.

[0017] The APD 116 executes commands and programs for selected functions, such as graphics operations and non-graphics operations that 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. In some examples, these compute processing operations are performed by executing compute shaders on the SIMD units 138.

[0018] The APD 116 includes compute units 132 that include one or more SIMD units 138 that are configured to perform operations at the request of the processor 102 (or another unit) 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 an individual lane, 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.

[0019] The basic unit of execution in compute units 132 is a work-item. Each work-item represents a single instantiation of a program that is to be executed in parallel in a particular lane. Work-items can be executed simultaneously (or partially simultaneously and partially sequentially) as a "wavefront" on a single SIMD processing unit 138. One or more wavefronts are 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. In alternatives, the wavefronts are executed on a single SIMD unit 138 or on different SIMD units 138. Wavefronts can be thought of as the largest collection of work-items that can be executed simultaneously (or pseudo-simultaneously) on a single SIMD unit 138. "Pseudo-simultaneous" execution occurs in the case of a wavefront that is larger than the number of lanes in a SIMD unit 138. In such a situation, wavefronts are executed over multiple cycles, with different collections of the work-items being executed in different cycles. An APD scheduler 136 is configured to perform operations related to

scheduling various workgroups and wavefronts on compute units 132 and SIMD units 138.

[0020] The parallelism afforded by the compute units 132 is suitable for graphics related operations such as pixel value calculations, vertex transformations, and other graphics operations. Thus in some instances, a graphics pipeline 134, which accepts graphics processing commands from the processor 102, provides computation tasks to the compute units 132 for execution in parallel.

[0021] The compute units 132 are also used to perform computation tasks not related to graphics or not performed as part of the "normal" operation of a graphics pipeline 134 (e.g., custom operations performed to supplement processing performed for operation of the graphics pipeline 134). An application 126 or other software executing on the processor 102 transmits programs that define such computation tasks to the APD 116 for execution.

[0022] 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 of the graphics processing pipeline 134. Each stage is implemented partially or fully as shader programs executing in the programmable compute units 132, or partially or fully as fixed-function, non-programmable hardware external to the compute units 132.

[0023] 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.

[0024] 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 include various

operations to transform the coordinates of the vertices. These operations include one or more of modeling transformations, viewing transformations, projection transformations, perspective division, and viewport transformations, which modify vertex coordinates, and other operations that modify non-coordinate attributes.

[0025] The vertex shader stage 304 is implemented partially or fully as vertex shader programs to be executed on one or more compute units 132. The vertex shader programs are provided by the processor 102 and are based on 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 compute units 132.

[0026] 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. The tessellator stage 308 generates a set of samples for the patch. The domain shader stage 310 calculates vertex positions for the vertices corresponding to the samples for the patch. The hull shader stage 306 and domain shader stage 310 can be implemented as shader programs to be executed on the compute units 132 that are compiled by the driver 122 as with the vertex shader stage 304.

[0027] The geometry shader stage 312 performs vertex operations on a primitive-by-primitive basis. 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. In some instances, a geometry shader program that is compiled by the driver 122 and that executes on the compute units 132 performs operations for the geometry shader stage 312.

[0028] The rasterizer stage 314 accepts and rasterizes simple primitives (triangles) generated upstream from the rasterizer stage 314. Rasterization

consists of determining which screen pixels (or sub-pixel samples) are covered by a particular primitive. Rasterization is performed by fixed function hardware.

[0029] The pixel shader stage 316 calculates output 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 pixel shader program that is compiled by the driver 122 and that executes on the compute units 132.

[0030] The output merger stage 318 accepts output from the pixel shader stage 316 and merges those outputs into a target surface, performing operations such as z-testing and alpha blending to determine the final color for the screen pixels. A target surface is the eventual target for a frame of the rendering operations within the graphics processing pipeline 134. The target surface may be at any location in memory (such as within a memory of the APD 116, or in memory 104).

[0031] The rasterizer stage 314 accepts triangles from earlier stages and performs scan conversion on the triangles to generate fragments. The fragments are data for individual pixels of a render target and include information such as location, depth, and coverage data, and later, after the pixel shader stage, shading data such as colors. The render target is the destination image to which rendering is occurring (i.e., colors or other values are being written). If the render target is a multi-sample image, then each pixel has multiple sample locations. The fragments that are generated by the rasterizer stage 314 are transmitted to the pixel shader stage 316, which determines color values for those fragments, and may determine other values as well.

[0032] Figure 4 illustrates a multi sample anti-aliasing load operation 400, according to an example. Anti-aliasing is a technique whereby data is generated for each of multiple samples of each pixel of a multi-sample render target. A multi-sample resolve operation, which can be performed in any manner (such as by software, hardware circuitry, or a combination thereof), but in some implementations is performed by a shader program executing on the compute units 132, down-samples the information of the multi-sample render target to

generate a full-resolution image. In an example where the multi-sample rate is 4x, the graphics processing pipeline 134 generates a multi-sample image having four samples per pixel. Then, a multi-sample resolve operation down-samples the multi-sample image to generate a full-resolution image. The number of samples for each pixel of the full-resolution image is one quarter of the number of samples of the multi-sample render target. The term "sample" includes one or more of color information (including one or more color components), depth information, and/or other information. In some examples, a sample in a multi-sample image has four color components and one depth component.

[0033] As described elsewhere, the compute units 132 include SIMD units 138 that perform operations in a single-instruction-multiple-data manner. More specifically, each active lane 402 executes the same instruction as all other lanes 402 in the SIMD unit 138 in any given clock cycle. Thus, when the SIMD unit 138 executes the multi-sample load operation, each active lane 402 of the SIMD unit 138 performs that load operation.

[0034] For any particular lane 402, the load operation involves fetching multiple elements 408 for one or more samples from a cache 404 into a vector register per lane 402. A number of different versions of the multi-sample load operation are disclosed herein. In general, these different versions vary based on what data is requested. In some examples, a single load operation for a single lane 402 loads the same color component (e.g., "R" for an "RGB" color scheme) for different samples into a single vector register 406 for the lane 402. In other examples, a single load operation for a single lane 402 loads multiple (such as all) color components for a single sample into a vector register 406 for that lane 402. In yet other examples, a single load operation for a single lane 402 loads one depth value for multiple samples into a vector register 406 for that lane 402. In any case, the elements 408 refer to individual data elements (such as color components or depth values) loaded by the load operation.

[0035] The illustrated cache 404 is a cache in a memory hierarchy from which the SIMD unit 138 fetches data into registers such as the vector register 406. In various examples, the cache 404 is within the SIMD unit 138, within a

compute unit 132 but outside of a SIMD unit 138, or within the APD 116 but outside of a compute unit 132. It should be understood that a miss in the cache 404 would result in the cache performing a cache line fill from higher up in the memory hierarchy.

[0036] To summary, the multi sample anti-alias load is an instruction that is executed by each active lane 402 of a SIMD unit 138. The instruction is performed at the request of a shader program executed on a SIMD unit 138. For any particular lane 402, the instruction specifies which elements of a multi-sample surface are loaded into a vector register 406 associated with that lane 402. In some examples, the elements loaded for one lane 402 are the same color component from different samples in the same pixel. In other examples, the elements are different color components from the same sample in a pixel. In other examples, the elements are depth values for different samples within the same pixel. A load instruction includes an indication of whether color components or depth components are loaded, as well as whether the same color component or different color components are loaded. Based on the sampling rate of the surface, and the organization of the stored data, the SIMD unit 138 selects a stride and loads the elements requested into a vector register 406 based on that stride.

[0037] In the situation that the load operation is to load the same component of different samples of a pixel, the number of samples of each pixel is, in some situations, different than the number of elements actually loaded by a particular load operation, where the number of elements loaded is, in some instances, determined by the size of the vector register 406 destination in comparison to the size of the color components. (For example, if the size of a vector register 406 is the same as the size of four elements, then a load operation loads for elements). In situations where the number of samples is lower than the number of elements to be loaded, the load operation loads the same component for different samples into a portion of the vector register 406. The load operation handles the remaining portion of the vector register 406 in any technically feasible manner, such as by repeating elements actually loaded into that portion, storing a constant into that portion, or placing any other value into that portion of the vector register 406. In

situations where the sampling rate is greater than the number of elements to be loaded, the load operation operations in two or more phases. In each phase, the same component for a different set of samples (e.g., samples 1-4, samples 5-8, etc.) is loaded. In some implementations, the load instruction includes a flag that indicates the phase (thus allowing a programmer or other author of a program (which is in some instances a shader program) to specify which sets of samples are loaded at any particular time. In other implementations, a single load instruction specifies two or more vector registers 406 and the SIMD unit 138 fetches the same color component for different sets of samples the data to those two or more vector registers 406. This "phasing" also applies to depth buffer loads for sampling rates higher than the vector register size, as illustrated, for example, in Figure 5C.

[0038] The load operation 400 is illustrated in the context of the SIMD unit 138 and in the context of SIMD processing, but operation of the load instruction outside of a SIMD context is contemplated in this disclosure as well.

[0039] Figures 5A-5C illustrate example variations for the multi-sample load instruction. Each of the operations illustrated in Figures 5A-5C assume a data format for the color buffer (the source from which the color components are loaded) where the color components for each individual sample are stored together, in one continuous chunk.

[0040] Figure 5A illustrates examples involving a multi-sample render target having 4 samples 502 per pixel 504. In a first example load operation – the color buffer load – same component operation 506(1), the SIMD unit 138 loads elements of the same color component for the pixel 504. As described above, based on the sample rate of the data from which the load is fetching elements, and on whether the load operation is for the same component or the same sample, the SIMD unit 138 selects a stride 508. The stride indicates 508 the number of elements in a cache data set 510 that are advanced over when obtaining the elements for the load operation. In Figure 5A, the number is 4. This is because in the cache data set 510(1) illustrated, the color data is arranged such that color components for the same pixel are in consecutive memory positions. Note that the color components are indicated with the notation SXCY, where X is the sample

number and Y is the component number. The stride 508 of 4 allows the load operation 506(1) to collect each of the same component for four different samples.

[0041] In a second example, color buffer load – same sample 506(2), the load has a stride of 1 due to the components for each sample being consecutive. This allows the load to obtain all components of a single sample 502 where, again, the color data is arranged as shown.

[0042] In a third example, depth buffer load 507(1), the stride is 1, because the depth data has only one component. Thus the depth buffer load 507(1) loads depth data for 4 different samples with a single instruction.

[0043] Figure 5B illustrates a different example mode of operation, where each pixel 524 has two samples 522 instead of 4 (i.e., the sampling rate is 2x). In this mode of operation, color buffer load operation – same component 526(1) has a stride of 4, since this operation obtains the same color component from different samples, and the format of the data being loaded is such that different color components of the same sample are consecutive in memory. However, because there are only two samples in a pixel, the load operation 526(1) obtains only two color components and not four. When writing data into the vector register 406, in various implementations, the load operation 526(1) repeats the two components of data, fills two elements of the vector register 406 with 0's or another constant, or places any other data into those two elements.

[0044] Color buffer load – same sample 526(2) loads consecutive data items. In the example shown, these consecutive data items are the four color components for a single sample 522. Depth buffer load operation 527(1) loads consecutive depth values, one for each sample, as illustrated.

[0045] Figure 5C illustrates another example, where each pixel has 8 samples 522. In this example, a color buffer load operation that loads the same number of elements into the same size vector register 406 as with the 4 sample and 2 sample examples is not able to load all of the same component for every sample in a pixel into such a vector register 406. Thus the load operation operates in two different phases. In the first phase, shown as color buffer load – same component, first phase 546(1), the load operation 546(1) loads the same color

component for a first four samples 522. In the second phase, shown as color buffer load same component, second phase 546(2), the load operation 546(2) loads the same color component for a second set of different samples. In the example illustrated, load operation 546(1) loads color component C1 for samples S1-S4 and load operation 546(2) loads color component C1 for samples S5-S8. Load operation 546(1) and load operation 546(2) each have a stride of 4, reflecting that four components are skipped over for each element obtained by the load operation 546, so that the same component for each sample is loaded.

[0046] For color buffer load – same sample 546(3), the load operation 546(3) loads with stride 1, meaning consecutive color components are loaded, as illustrated.

[0047] For the depth buffer load, two phases are illustrated. In one mode, depth buffer load, first phase 547(1), the load instruction 547(1) loads a depth component for a first set of samples, with stride 1. In another mode, depth buffer load, second phase 547(2), the load instruction 547(2) loads a depth component for a second set of samples, with stride 1. In the examples, the first set of samples are samples S1-S4 and the second set of samples are samples S5-S8.

[0048] Figure 5D illustrates a different data layout 560 than that shown in Figures 5A-5C. Instead of the data set being arranged such that all color components of a single sample are consecutive, in Figure 5D, the same color component for different samples are consecutive. It should be understood that the example of Figure 5D is for a 4 sample render target and that other sampling rates are of course possible.

[0049] The ordering of the data in the data set 560 of Figure 5D illustrates that the versions of the load operations illustrated in Figures 5A-5C performs different operations depending on how the data being loaded is organized. In Figure 5A, with the data as shown in that Figure, load 506(1), which operates with stride 4, loads the same color component of different samples. However, if the color load 506(1) were to operate with the data as shown in Figure 5D, then the load 506(1) would instead fetch four color components of a single sample (thus having stride 1). Similarly, load 506(2), which operates with stride 1, would fetch

the same component of a single sample when operated on for the data of Figure 5D, thus having stride 4.

[0050] Note that the various numbers of items (e.g., color components, samples per pixel, vector register size, and the like) illustrated in Figures 5A-5D are exemplary in nature, and that variations in one or more of these numbers are contemplated by the present disclosure. For example, in some variations, the number of components for sample colors is different than four. In some variations, the number of elements (components or depth values) that can fit into a vector register 406 is different than four. In some variations, sample rates other than 2x, 4x, or 8x are possible. Many other variations are possible. In some examples, the stride is other than four. For example, where there are five components per color and the components for each sample are consecutive in memory, a load of the same color component for different samples has a stride of five. In some examples, the stride in such a situation is equal to the number of components per color. In other examples where the load operation loads non-consecutive elements, the stride is set to match the spacing of those non-consecutive elements.

[0051] Note that the load operations described, in some situations, gain cache efficiency as compared with operations where individual samples or depth values are loaded from the cache one at a time. More specifically, it is possible for a cache line to be evicted in between load operations. Thus with more data being loaded at the same time, fewer cache evictions will occur as data is being loaded.

[0052] Figure 6 is a flow diagram of a method 600 for performing multi-sample anti-aliasing operations, according to an example. Although described with respect to Figures 1-5D, those of skill in the art will understand that any system, configured to perform the steps of method 600 in any technically feasible order, falls within the scope of the present disclosure.

[0053] The method 600 begins at step 602, where a processor such as a SIMD unit 138 detects a multi-sample anti-aliasing load instruction. In various examples, the load instruction is part of an instruction set architecture for a processor such as the SIMD units 138. More specifically, the SIMD units 138

execute shader programs that include instructions, some of which are the multi-sample anti-aliasing load instructions.

[0054] The multi-sample load operation specifies which buffer to load from, where the term "buffer" means a portion of memory storing data to be loaded. In some examples, the buffer is a render target storing color data or storing depth data. In some examples, the load operation explicitly specifies whether the data of the buffer is color data or depth data. In other examples, the SIMD unit 138 determines, by examining the buffer itself, or metadata for the buffer, whether the data of the buffer is color data or depth data.

[0055] At step 604, the SIMD unit 138 determines the sampling rate of the source data for the load operation, the data storage format of the source data if color data is being loaded, and the loading mode indicating whether the load operation requests the same color components of different samples, different color components of the same sample, or depth data. The sampling rate is the number of samples per pixel. The data storage format indicates whether different color components of the same sample are consecutive or the same component of different samples are consecutive.

[0056] At step 606, based on the information determined at step 604 the SIMD unit 138 loads the data requested by the load operation. More specifically, the SIMD unit 138 selects a stride based on that information, obtains the data elements based on the characteristics of the operation determined at step 604, and obtains the elements from memory based on the stride. Technique for performing such a load based on this information are described above with respect to Figures 4-5D.

[0057] Once loaded, the data is used in any technically feasible manner. In one example, a shader program executing in the SIMD unit 138 performs a resolve operation, generating a lower resolution image from the multi-sample images. In an example, each work-item (which corresponds to a lane 402) of the shader program resolves the four samples loaded by the load operation into a single sample. Although any technique may be used, one example technique involves averaging the values loaded.

[0058] Each of the functional units illustrated in the figures represent hardware circuitry configured to perform the operations described herein, software configured to perform the operations described herein, or a combination of software and hardware configured to perform the steps described herein. A non-exclusive list of such units includes the storage 106, the processor 102, the output driver 114, the APD 116, the memory 104, the input driver 112, the input devices 106, the output devices 110, the display device 118, the operating system 120, the driver 122, the applications 126, the APD scheduler 136, the graphics processing pipeline 134, the compute units 132, the SIMD units 138, any of the stages of the graphics processing pipeline 134, the lanes 402 of the SIMD unit 138, the cache 404, and the vector registers 406.

[0059] 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 can be used alone without the other features and elements or in various combinations with or without other features and elements.

[0060] The methods provided can 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 can 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 can be maskworks that are then used in a semiconductor manufacturing process to manufacture a processor which implements features of the disclosure.

[0061] The methods or flow charts provided herein can 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 for performing multi-sample anti-aliasing operations, the method comprising:

detecting an instruction for a multi-sample anti-aliasing load operation;

determining a sampling rate of source data for the load operation, a data storage format of the source data, and a loading mode for the load operation, wherein the loading mode indicates whether the load operation requests same color components, different color components, or depth data; and

based on the determined sampling rate, data storage format, and loading mode, loading data from a multi-sample source into a register.

2. The method of claim 1, wherein the data storage format indicates different color components of samples are consecutive.

3. The method of claim 2, wherein:

the loading mode indicates that the load operation requests the same color components; and

loading the data from the multi-sample source includes loading with a stride that is greater than 1.

4. The method of claim 2, wherein:

the loading mode indicates that the load operation requests different color components of the same sample; and

loading the data from the multi-sample source includes loading with a stride of 1.

5. The method of claim 1, wherein the data storage format indicates the same color component of different samples are consecutive.

6. The method of claim 5, wherein:

the loading mode indicates that the load operation requests the same color components; and

loading the data from the multi-sample source includes loading with a stride of 1.

7. The method of claim 5, wherein:

the loading mode indicates that the load operation requests different color components of the same sample; and

loading the data from the multi-sample source includes loading with a stride that is greater than 1.

8. The method of claim 1, wherein:

a sampling rate of the source data is greater than a number of elements that fit into the register; and

loading the data includes loading less than all samples of a pixel into the register.

9. The method of claim 1, wherein:

a sampling rate of the source data is less than a number of elements that fit into the register; and

loading the data includes repeating loaded data in the register.

10. A system for performing multi-sample anti-aliasing operations, the system comprising:

a register; and

a processor configured to:

detect an instruction for a multi-sample anti-aliasing load operation;

determine a sampling rate of source data for the load operation, a data storage format of the source data, and a loading mode for the load operation,

wherein the loading mode indicates whether the load operation requests same color components, different color components, or depth data; and

based on the determined sampling rate, data storage format, and loading mode, load data from a multi-sample source into a register.

11. The system of claim 10, wherein the data storage format indicates different color components of samples are consecutive.

12. The system of claim 11, wherein:

the loading mode indicates that the load operation requests the same color components; and

loading the data from the multi-sample source includes loading with a stride that is greater than 1.

13. The system of claim 11, wherein:

the loading mode indicates that the load operation requests different color components of the same sample; and

loading the data from the multi-sample source includes loading with a stride of 1.

14. The system of claim 10, wherein the data storage format indicates the same color component of different samples are consecutive.

15. The system of claim 14, wherein:

the loading mode indicates that the load operation requests the same color components; and

loading the data from the multi-sample source includes loading with a stride of 1.

16. The system of claim 14, wherein:

the loading mode indicates that the load operation requests different color components of the same sample; and

loading the data from the multi-sample source includes loading with a stride that is greater than 1.

17. The system of claim 10, wherein:

a sampling rate of the source data is greater than a number of elements that fit into the register; and

loading the data includes loading less than all samples of a pixel into the register.

18. The system of claim 10, wherein:

a sampling rate of the source data is less than a number of elements that fit into the register; and

loading the data includes repeating loaded data in the register.

19. An accelerated processing device for performing multi-sample anti-aliasing operations, the accelerated processing device comprising:

a register; and

a single instruction multiple data processing unit configured to:

detect an instruction for a multi-sample anti-aliasing load operation;

determine a sampling rate of source data for the load operation, a data storage format of the source data, and a loading mode for the load operation, wherein the loading mode indicates whether the load operation requests same color components, different color components, or depth data; and

based on the determined sampling rate, data storage format, and loading mode, load data from a multi-sample source into a register.

20. The accelerated processing device of claim 19, wherein:

the data storage format indicates different color components of samples are consecutive, or the data storage format indicates the same color component of different samples are consecutive.

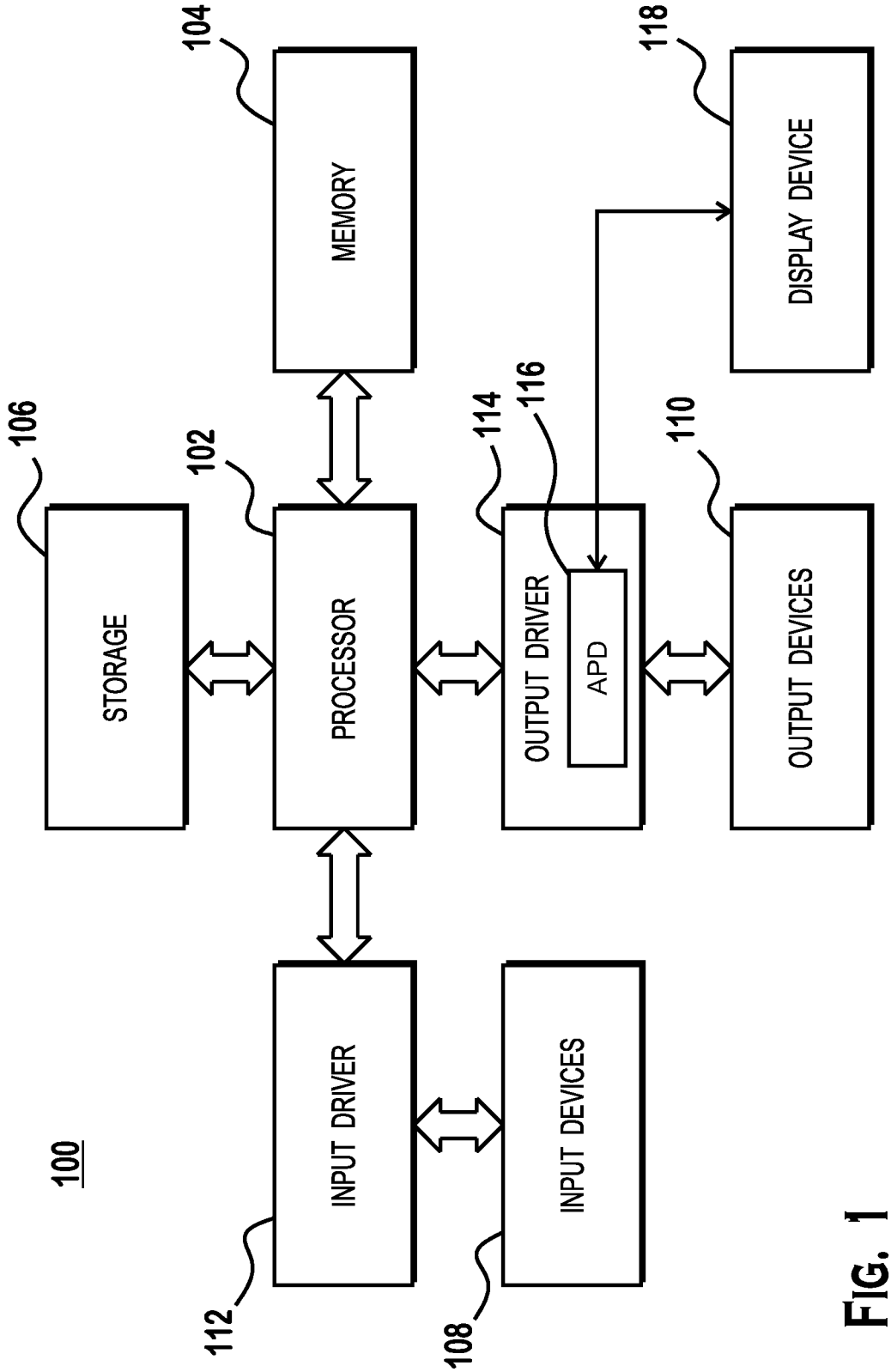
100
104
MEMORY
118
DISPLAY DEVICE
106
102
114
116
110
STORAGE
PROCESSOR
OUTPUT DRIVER
APD
OUTPUT DEVICES
INPUT DRIVER
INPUT DEVICES
112
108

FIG. 1

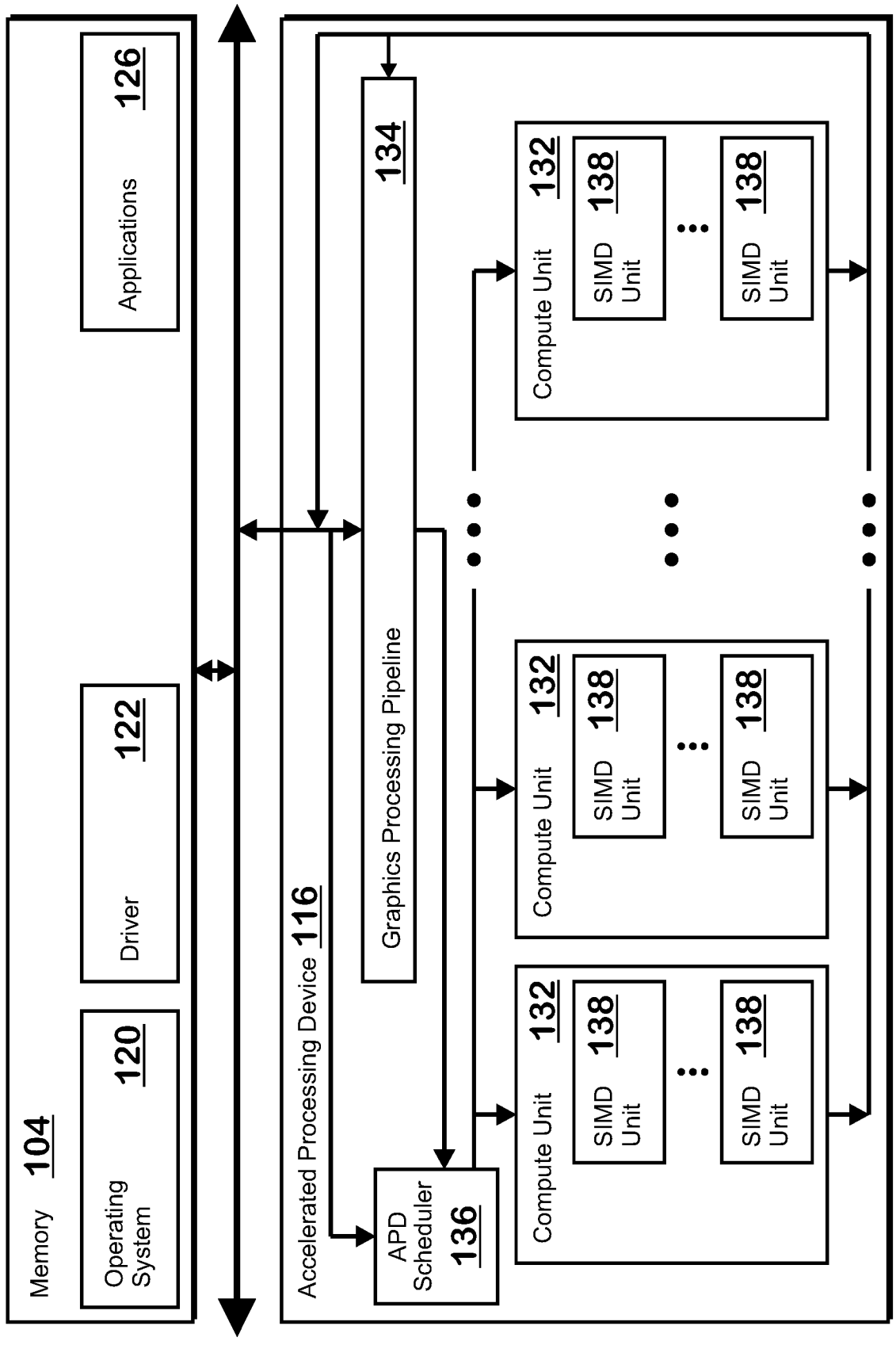
Memory 104
Operating System 120
Driver 122
Applications 126
Accelerated Processing Device 116
Graphics Processing Pipeline 134
APD Scheduler 136
Compute Unit 132
SIMD Unit 138

FIG. 2

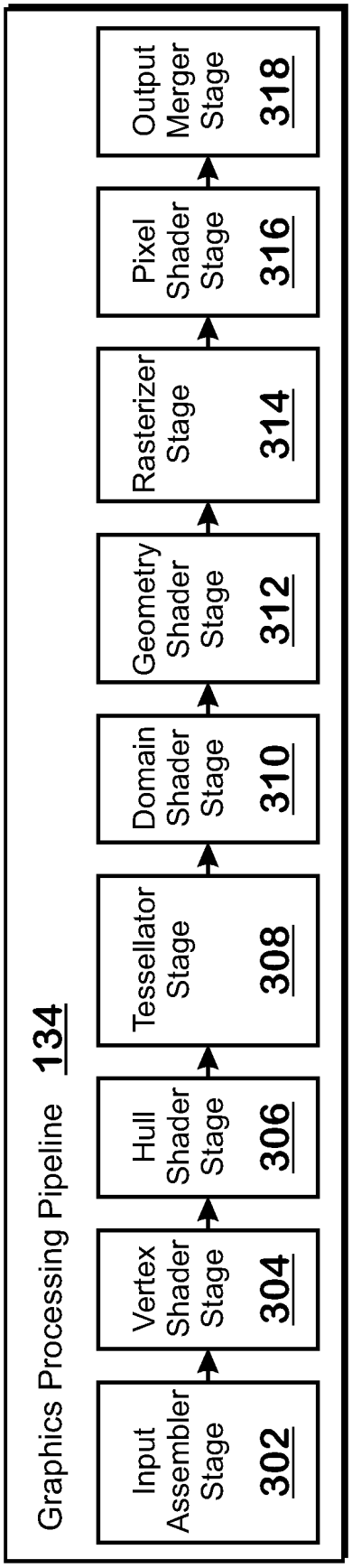
Graphics Processing Pipeline 134
Input Assembler Stage 302
Vertex Shader Stage 304
Hull Shader Stage 306
Tessellator Stage 308
Domain Shader Stage 310
Geometry Shader Stage 312
Rasterizer Stage 314
Pixel Shader Stage 316
Output Merger Stage 318

FIG. 3

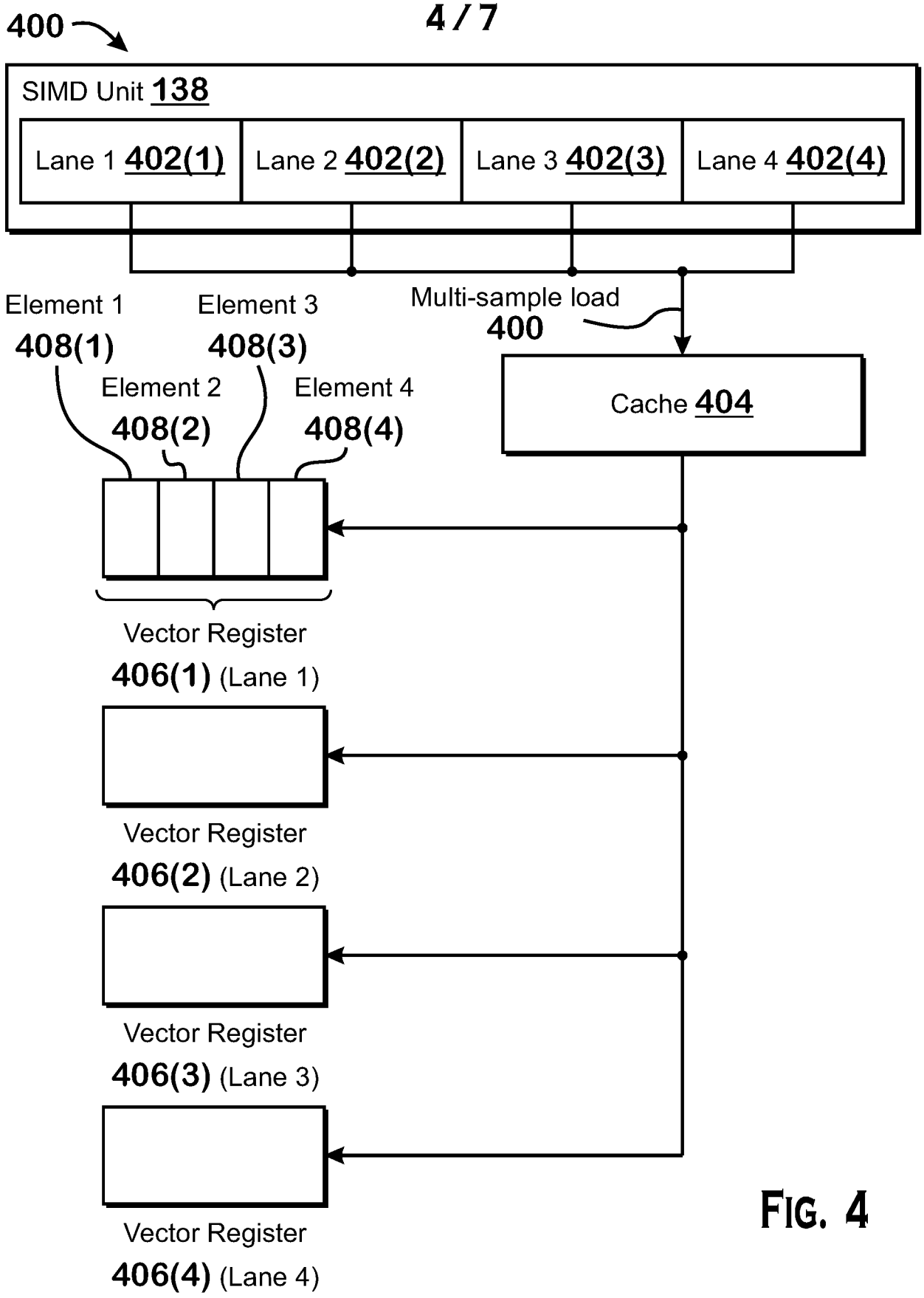
400
SIMD Unit 138
Lane 1 402(1)
Lane 2 402(2)
Lane 3 402(3)
Lane 4 402(4)
Element 1
408(1)
Element 3
408(3)
Element 2
408(2)
Element 4
408(4)
Multi-sample load
400
Cache 404
Vector Register
406(1) (Lane 1)
Vector Register
406(2) (Lane 2)
Vector Register
406(3) (Lane 3)
Vector Register
406(4) (Lane 4)

FIG. 4

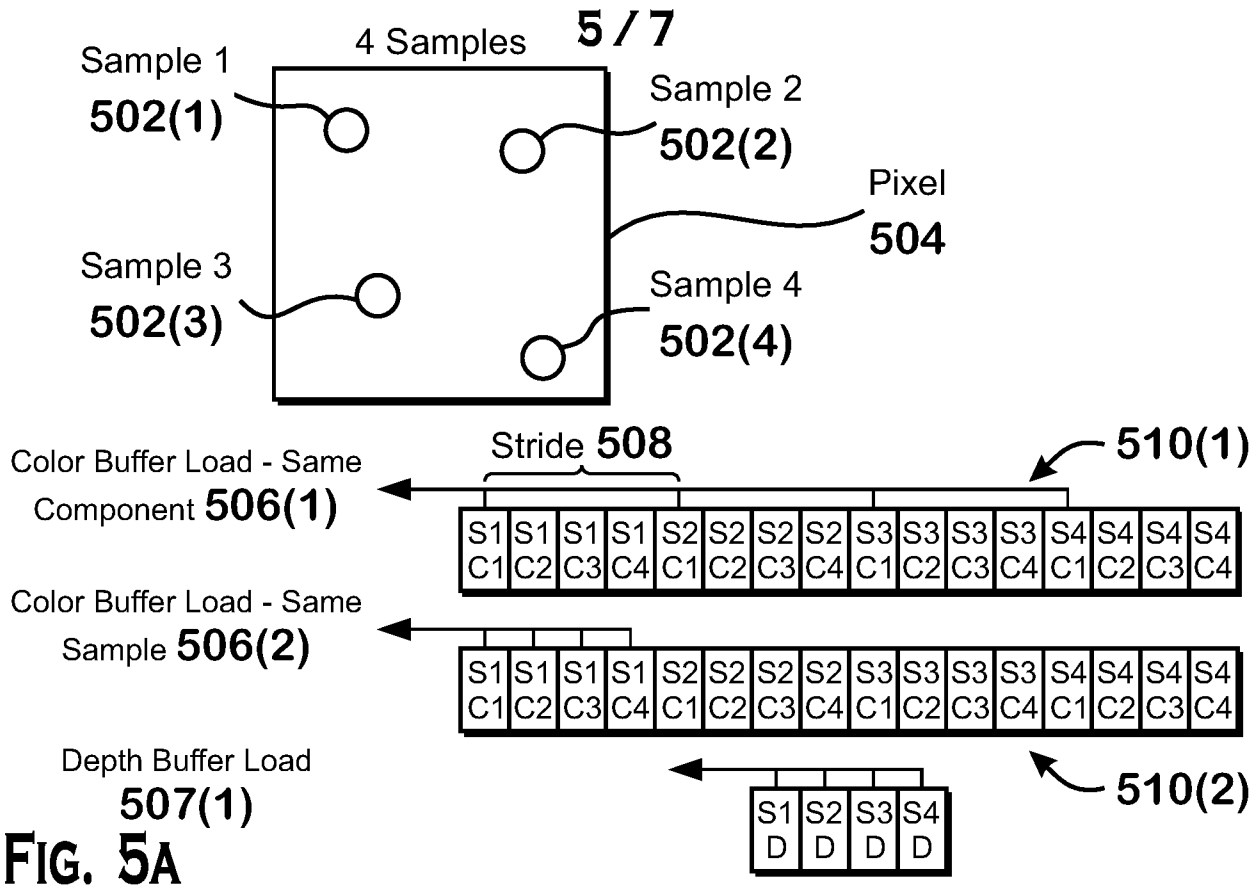
4 Samples
Sample 1
502(1)
Sample 2
502(2)
Pixel
504
Sample 3
502(3)
Sample 4
502(4)
Stride 508
510(1)
Color Buffer Load - Same Component 506(1)
Color Buffer Load - Same Sample 506(2)
S1 C1 S1 C2 S1 C3 S1 C4 S2 C1 S2 C2 S2 C3 S2 C4 S3 C1 S3 C2 S3 C3 S3 C4 S4 C1 S4 C2 S4 C3 S4 C4
Depth Buffer Load
507(1)
S1 D S2 D S3 D S4 D
510(2)

FIG. 5A

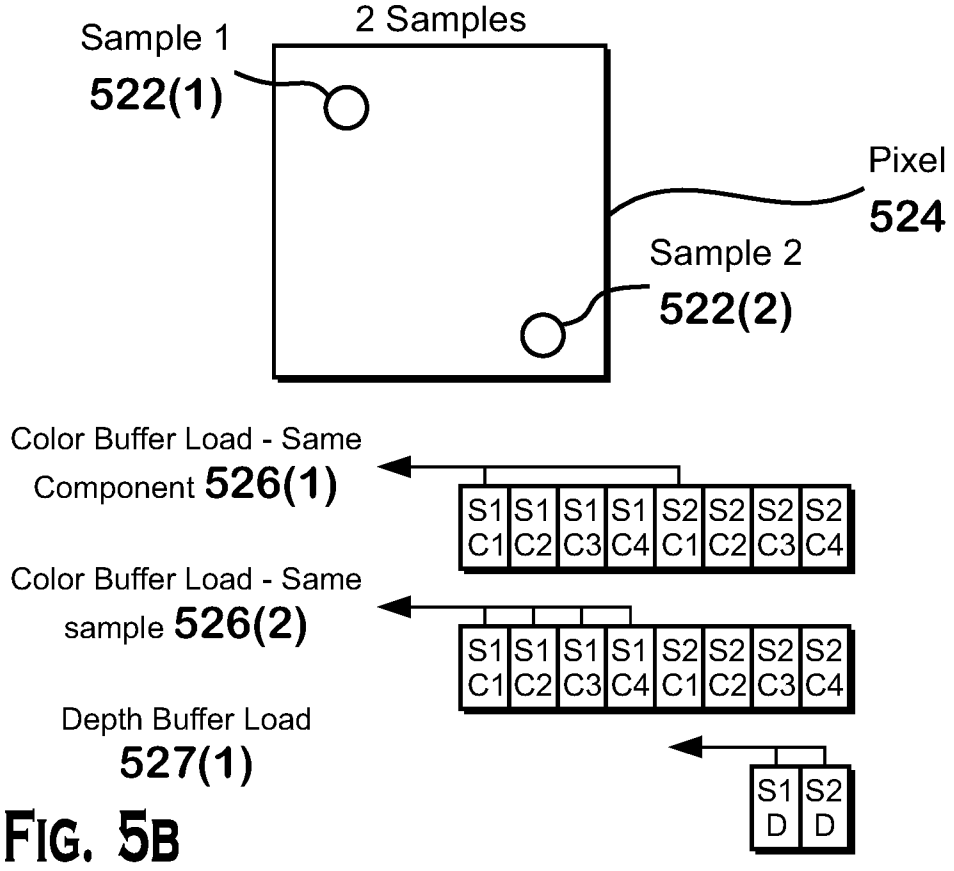
2 Samples
Sample 1
522(1)
Pixel
524
Sample 2
522(2)
Color Buffer Load - Same Component 526(1)
Color Buffer Load - Same sample 526(2)
S1 C1 S1 C2 S1 C3 S1 C4 S2 C1 S2 C2 S2 C3 S2 C4
Depth Buffer Load
527(1)
S1 D S2 D

FIG. 5B

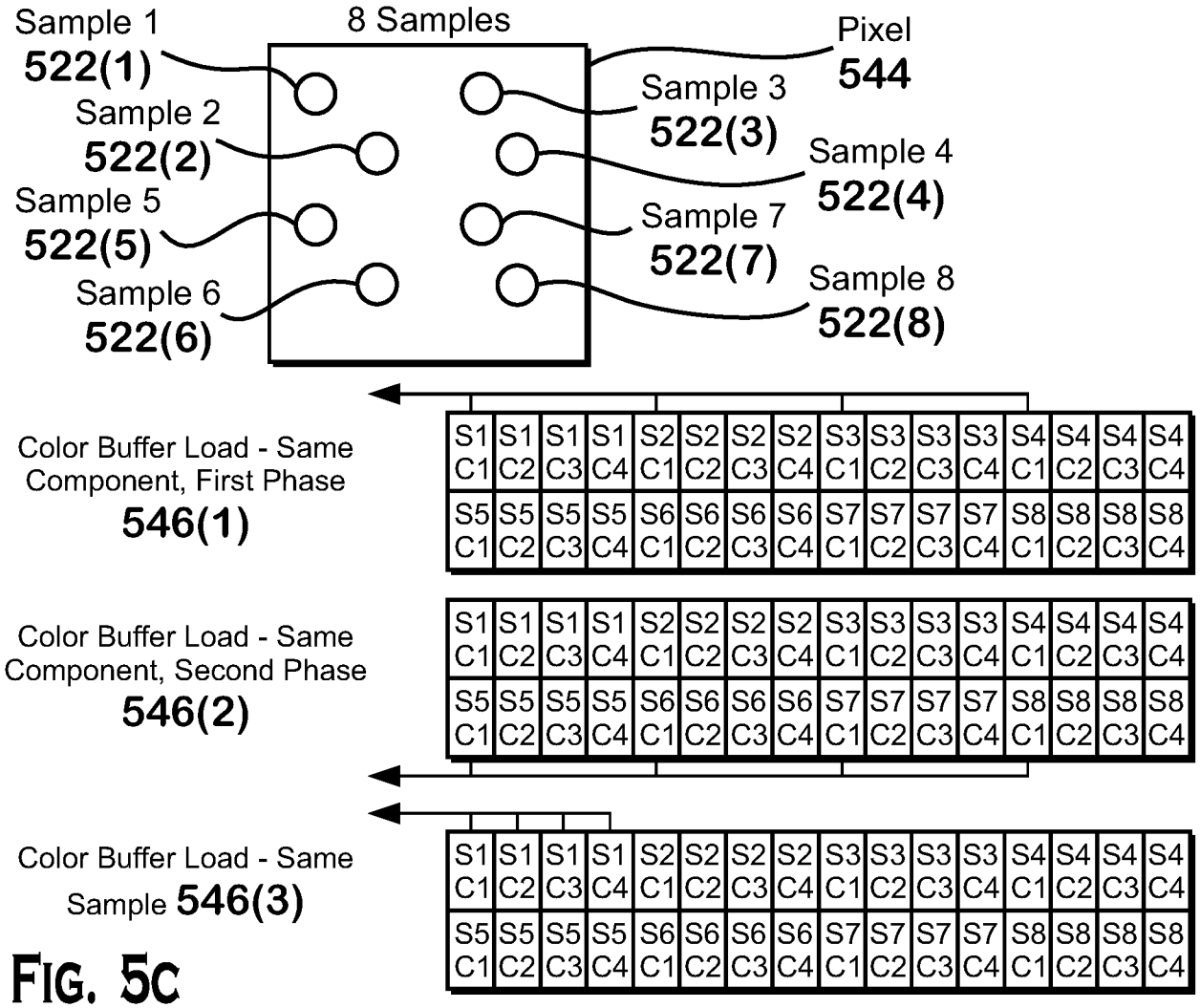
Sample 1
522(1)
8 Samples
Pixel
544
Sample 2
522(2)
Sample 3
522(3)
Sample 4
522(4)
Sample 5
522(5)
Sample 7
522(7)
Sample 6
522(6)
Sample 8
522(8)
Color Buffer Load - Same Component, First Phase
546(1)
Color Buffer Load - Same Component, Second Phase
546(2)
Color Buffer Load - Same Sample 546(3)

FIG. 5C

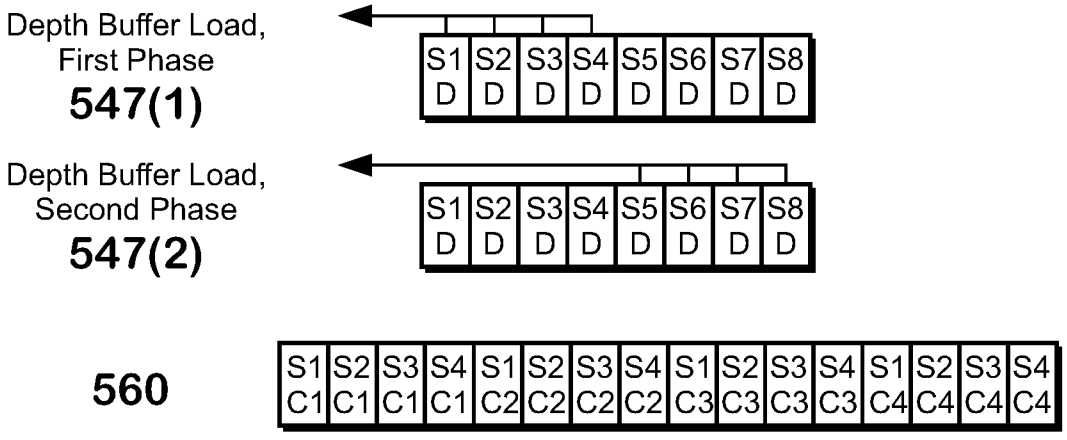
Depth Buffer Load, First Phase
547(1)
Depth Buffer Load, Second Phase
547(2)
560

FIG. 5D

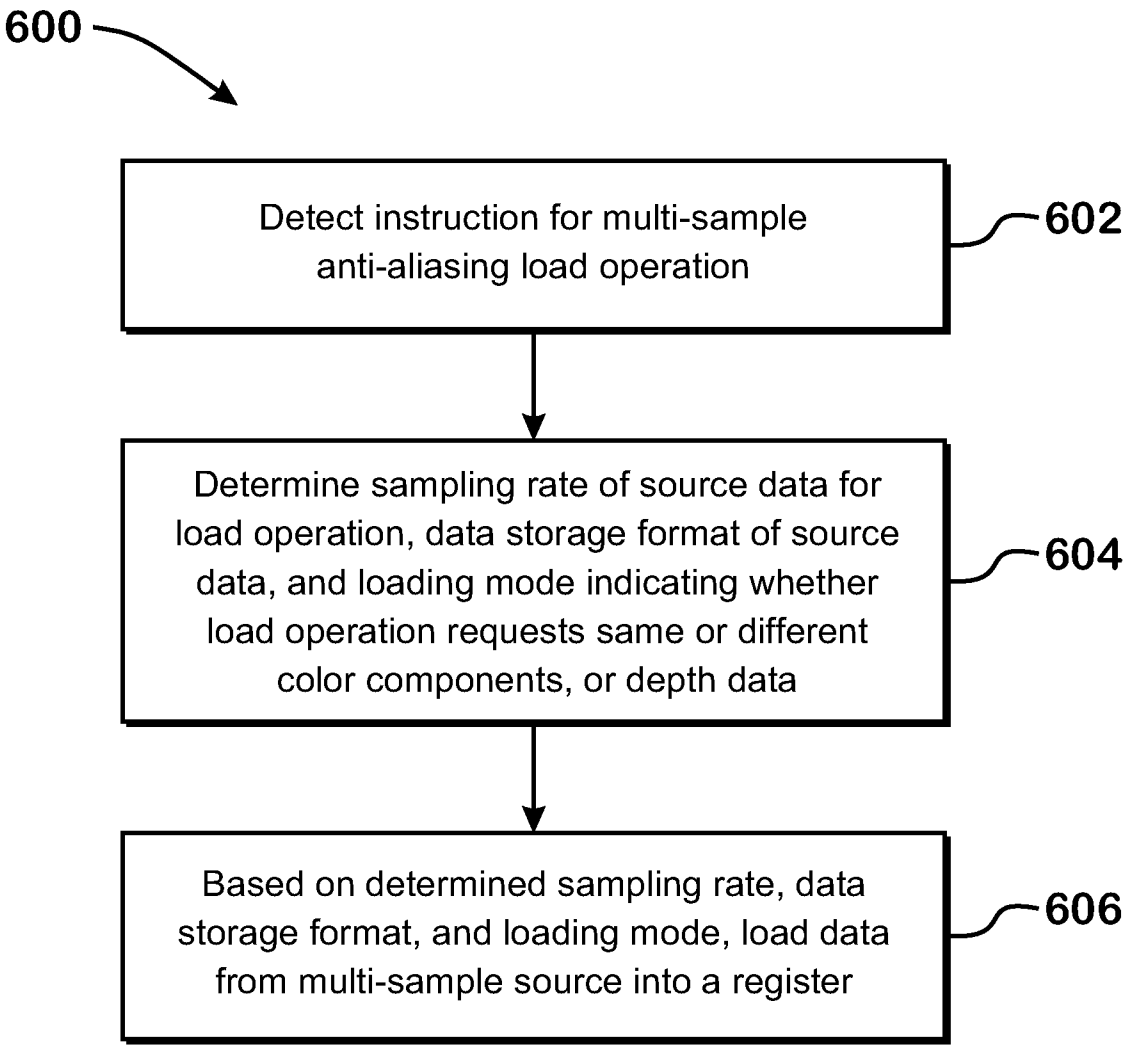
600
Detect instruction for multi-sample anti-aliasing load operation
602
Determine sampling rate of source data for load operation, data storage format of source data, and loading mode indicating whether load operation requests same or different color components, or depth data
604
Based on determined sampling rate, data storage format, and loading mode, load data from multi-sample source into a register
606

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2021/036485

A. CLASSIFICATION OF SUBJECT MATTER

G06T 15/00(2006.01)i; **G06T 15/04**(2011.01)i; **G06T 15/40**(2011.01)i; **G06T 7/50**(2017.01)i; **G06T 7/90**(2017.01)i

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 15/00(2006.01); G06T 1/20(2006.01); G06T 1/60(2006.01); G06T 15/50(2006.01); G06T 7/13(2017.01); G06T 9/00(2006.01)

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

Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & keywords: multi-sample anti-aliasing, load, operation, color, component, register

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2018-0182058 A1 (APPLE INC.) 28 June 2018 (2018-06-28) paragraphs [0043], [0091]-[0095], [0101]-[0102], [0109]; and figures 6-7	1-20
Y	US 2015-0091918 A1 (SAMSUNG ELECTRONICS CO., LTD.) 02 April 2015 (2015-04-02) paragraphs [0105], [0115]-[0116], [0120]; and figures 10A-14C	1-20
A	US 2018-0308211 A1 (INTEL CORPORATION) 25 October 2018 (2018-10-25) claims 1-10	1-20
A	US 2019-0392631 A1 (FILIP STRUGAR) 26 December 2019 (2019-12-26) claims 1-12	1-20
A	US 10223809 B2 (INTEL CORPORATION) 05 March 2019 (2019-03-05) claims 1-7; and figure 3	1-20

☐ Further documents are listed in the continuation of Box C. ☑ 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
"D" document cited by the applicant in the international application
"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
24 September 2021	**27 September 2021**

Name and mailing address of the ISA/KR	Authorized officer
Korean Intellectual Property Office **189 Cheongsa-ro, Seo-gu, Daejeon** **35208, Republic of Korea**	**YANG, JEONG ROK**
Facsimile No. **+82-42-481-8578**	Telephone No. **+82-42-481-5709**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/US2021/036485

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2018-0182058 A1	28 June 2018	CN 109964244 A	02 July 2019
		US 10445852 B2	15 October 2019
		WO 2018-118363 A1	28 June 2018
US 2015-0091918 A1	02 April 2015	KR 10-2015-0035161 A	06 April 2015
		KR 10-2152735 B1	21 September 2020
		US 9207936 B2	08 December 2015
US 2018-0308211 A1	25 October 2018	US 10297047 B2	21 May 2019
		US 10810764 B2	20 October 2020
		US 2019-0244393 A1	08 August 2019
		US 2021-0118187 A1	22 April 2021
US 2019-0392631 A1	26 December 2019	CN 110634106 A	31 December 2019
		KR 10-2019-0143803 A	31 December 2019
		US 10902605 B2	26 January 2021
US 10223809 B2	05 March 2019	US 2017-0345186 A1	30 November 2017
		WO 2017-205005 A1	30 November 2017