



- (51) **International Patent Classification:**
H04N 19/60 (2014.01) *H04N 19/00* (2014.01)
- (21) **International Application Number:**
PCT/US2015/056675
- (22) **International Filing Date:**
21 October 2015 (21.10.2015)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
14/552,665 25 November 2014 (25.11.2014) US
- (71) **Applicant:** INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).
- (72) **Inventors:** BHATTACHARJEE, Susanta; Intel Technology India Pvt. Ltd. #23-56P, Devarabeesanahalli, Outer Ring Road, Varthur Hobli, Bangalore, KA 560103 (IN). KP, Sameer; #G18, Richfields Apt, Outer Ring Road, Marathahalli, Bangalore, KA 560037 (IN).
- (74) **Agents:** TROP, Timothy N. et al.; Trop, Pruner & Hu, P.C., 1616 S. Voss Rd., Ste. 750, Houston, Texas 77057-2631 (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, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT,

HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, 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, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

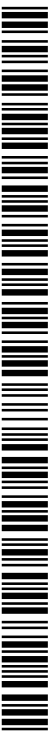
- (84) **Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, 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).

Declarations under Rule 4.17:

- as to the identity of the inventor (Rule 4.17(i))
- as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))
- as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))

Published:

- with international search report (Art. 21(3))
- before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))



WO 2016/085589 A1

(54) **Title:** COMPRESSING THE SIZE OF COLOR LOOKUP TABLES

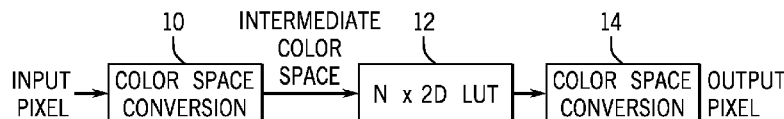


FIG. 1

(57) **Abstract:** By converting a first color space to a second color space, using a two-dimensional lookup table in said second color space, and converting from said second color space to said first color space, it may be possible to use one or more two-dimensional lookup tables (LUTs) to do a task conventionally handled by three-dimensional lookup tables. This may reduce storage requirements and memory bandwidth requirements in some embodiments. In general a color pixel with N color components can be processed with n number of M dimensional LUT where M<N and n is some chosen positive integer number.

COMPRESSING THE SIZE OF COLOR LOOKUP TABLES

Background

[0001] This relates generally to graphics processing.

[0002] In a variety of different circumstances, the colors that will be shown on a display need to be adjusted. Examples of such adjustments includes skin tone adjustments, color temperature adjustments, and color saturation adjustments, to mention a few examples.

[0003] Typically these color transformations are implemented using a three-dimensional lookup table. The problem with three-dimensional lookup tables is that the dimensions of the lookup table are a function of the number of input color components in the chosen color space. As an example, a lookup table for sRGB color space requires three inputs and hence uses a three-dimensional lookup table. The size of such a lookup table is 48 megabytes when both input and output are in the sRGB space with a depth of 8 bits per color.

[0004] Thus the use of three-dimensional lookup tables involves an enormous burden in terms of memory capacity and memory bandwidth requirements.

Brief Description Of The Drawings

[0005] Some embodiments are described with respect to the following figures:

Figure 1 is a process flow for one embodiment to the present invention;

Figure 2 is a flow chart for one embodiment to the present invention;

Figure 3 is a schematic depiction of one embodiment;

Figure 4 is a schematic depiction of another embodiment;

Figure 5 is a schematic depiction of still another embodiment;

Figure 6 is a block diagram of a data processing system according to one embodiment;

Figure 7 is a block diagram of the processor shown in Figure 6 according to one embodiment;

Figure 8 is a block diagram of the graphics processor of Figure 6 according to one embodiment.

Figure 9 is a block diagram of a graphics processing engine according to one embodiment;

Figure 10 is a block diagram of a graphics processor according to another embodiment;

Figure 11 illustrates thread execution logic for one embodiment;

Figure 12 is a block diagram of a graphics processor execution unit instruction format according to one embodiment;

Figure 13 is a block diagram of another embodiment of a graphics processor;

Figure 14A is a block diagram of a graphics processor command format according to one embodiment;

Figure 14B is a block diagram of a graphics processor command sequence according to one embodiment; and

Figure 15 is a graphics software architecture for one embodiment.

Detailed Description

[0006] By converting a first color space to a second color space, it may be possible to use one or more two-dimensional lookup tables (LUTs) to do a task conventionally handled by three-dimensional lookup tables. This may reduce storage requirements and memory bandwidth requirements in some embodiments.

[0007] In general, in any case where a given color component such as brightness does not change during the color transformation algorithm being implemented, it is possible to convert to an intermediate color space that only requires two-dimensional lookup tables. For example, if color information is in sRGB and the transformation involves only chrominance (Cb and Cr) color components and not the brightness

(luma) or Y color component, by transforming from sRGB to YCbCr color space, a single lookup table may be possible, in some embodiments, that only involves the Cb and Cr color components.

[0008] As a result, a two-dimensional lookup table may be used. Then after the transformation has been completed, the color space may be converted back to the first color space.

[0009] The net result may be to significantly reduce the size of a lookup table at the lesser cost of two extra color conversion steps. However extra mathematical operations, done to color space conversion, may be compensated for by less computation during interpolation of lookup table data of a lower dimension lookup table.

[0010] The following table shows a comparison of lookup table sizes between traditional three-dimensional 8 bit RGB space and a Nx2D space with an example value for the number of 2D lookup tables, N=3. Accuracy of the color transformation increases with the number of samples in the lookup table:

NUMBER OF LUT SAMPLES	TRADITIONAL (3D RGB)	Proposed (3x2D) LUT SIZE IN BYTES
9	2.1K	486
17	14.3K	1734
33	105K	6.38K
65	805K	24.76K
129	6.1M	97.5K
256	48.0M	384K

[0011] The table above shows the savings that can be achieved in storage requirements in some embodiments. As an example, a skin tone enhancement algorithm usually implemented with a three-dimensional lookup table may be

implemented using YCbCr as intermediate color space with three lookup tables (N=3) by converting from RGB color space to YCbCr color space, using three two-dimensional lookup tables and then converting back to RGB color space.

[0012] Referring to **Figure 1**, the general steps are to receive an input pixel and then to convert to an appropriate color space as indicated in block 10. Then the intermediate color space is used (block 12) with only two-dimensional lookup tables of the required number. Finally after the color conversion has occurred, the intermediate color space can be transformed back to the original color space as indicated in block 14.

[0013] A sequence for creating the lookup table is shown in **Figure 2**. The sequence may be implemented in software, firmware and/or hardware. In software and firmware embodiments it may be implemented by computer executed instructions stored in one or more non-transitory computer readable media such as magnetic, optical or semiconductor storage.

[0014] Initially, the color transformation algorithm is selected as indicated in block 16. This could include an algorithm for gamut mapping, color correction, adaptive brightness, adaptive contrast enhancement, skin tone adjustment, color temperature or whiteness adjustments, to mention a few examples. Then the intermediate color space is chosen as indicated in block 18. For example, an intermediate color space may be chosen so that the color space is one in which only two of three color space components are actually utilized in a color transformation facilitating the transition to one or more two-dimensional lookup tables.

[0015] Then the color components that are needed for the two-dimensional lookup table are selected as indicated in block 20. Next the number of lookup tables are chosen as indicated in block 22. Finally the lookup tables of the required number using the chosen algorithm are created as indicated in block 24.

[0016] There may be cases when multiple lookup tables may be needed. For example in the skin tone adjustment, one algorithm may be used for darker skin tones and another algorithm may be used for correcting lighter skin tones. Thus an RGB input pixel is converted to YCbCr color space and then two lookup tables are

used to adjust the Cb and Cr components, one lookup table being provided for the darker skin tones and another lookup table being provided for the lighter skin tones. In some cases, a detector may be needed to assess whether light or darker skin tones are implicated.

[0017] The following example describes a lookup table creation process for an algorithm that enhances color saturation of a pixel if its hue is within a specified range. Initially the input and output color space is noted and in this case is sRGB. The chosen intermediate color space is YCbCr. The transformation algorithm is to detect hue and transform saturation if hue is within a set of given ranges. The number of lookup tables in this example is 1. The number of samples in a lookup table is 33. The created lookup table size with an 8 bit color depth is 2178 bytes. The lookup table size with the traditional three-dimensional lookup table is 105 kilobytes.

[0018] Thus in accordance with one embodiment, a hardware embodiment, shown in **Figure 3**, may implement a two-dimensional lookup table technique for compensating for skin color darkness. Initially, the color information is passed to a color space conversion unit 30. In this example, the color space may be converted from RGB to YCbCr. Then a skin darkness detector 32 detects whether skin tones are implicated and if so how dark the skin colors are and selects the appropriate lookup table, either table 34 or table 36, depending on whether the skin colors are darker than a threshold or less dark than a threshold. Then the appropriate two-dimensional lookup table is used based on the determination of skin darkness. Finally, the color space is converted back to the original color space at color space conversion unit 38.

[0019] In accordance with another embodiment, partial hue and saturation control can be done using a two-dimensional lookup table instead of a three-dimensional lookup table. The hue of a pixel is detected and the hue/saturation is adjusted as required by the user. The user may provide hue/saturation adjustment factors for a few anchor colors (hues), in one embodiment, six anchor pixels. Then the user input can be taken through a slider control on a graphical user interface in one embodiment. The algorithm converts the RGB pixel to a YCbCr pixel and alters the

Cb and Cr components only. Hence a single two-dimensional lookup table is sufficient.

[0020] Referring to **Figure 4**, a hardware embodiment may be implemented by a hue detect module 40 that receives an RGB pixel. After detecting the hue, the user hue/saturation adjustment factors are received as indicated in block 42, for example from a graphical user interface. Then the color space is converted to YCbCr as indicated in block 44. Next the Cb and Cr components are adjusted using a two-dimensional lookup table (LUT) based on the user adjustment factors, as indicated in block 46. Finally, the color space is converted back to RGB (block 48).

[0021] In accordance with still another embodiment, a gamut compression algorithm may be implemented. The algorithm maps out gamut pixels within a gamut of a particular display panel. It converts an RGB pixel to the HSV color space and modifies only the S and V components of that color space, keeping the H component unchanged. Although the S and V adjustment involves an H component as an input, the entire hue region can be divided into three to six sub-regions and those many two-dimensional lookup tables with corresponding S and V components can be utilized.

[0022] Referring to **Figure 5**, a hardware embodiment may be implemented by a series of modules. The first module 50 detects an out of gamut pixel. Then as shown in block 52, the color space is converted from RGB to HSV. A hue region is divided into N sub-regions, as indicated in block 54. Next N two-dimensional LUTs are selected (based on the number of sub-regions) and then the S and V components are modified (block 56). Finally, the color space is converted back to RGB as indicated in block 60 after modifying the S and V components.

[0023] The embodiments of Figures 4 and 5 may also be implemented in software or firmware.

[0024] While the RGBW color space is used in some displays, color spaces with more color components will be available. The principles described herein can be used with more color components. A color pixel with N color components can be

processed with n number of M dimensional LUT where $M < N$ and n is some chosen positive integer number.

[0025] **Figure 6** is a block diagram of a data processing system 100, according to an embodiment. The data processing system 100 includes one or more processors 102 and one or more graphics processors 108, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 102 or processor cores 107. In one embodiment, the data processing system 100 is a system on a chip integrated circuit (SOC) for use in mobile, handheld, or embedded devices.

[0026] An embodiment of the data processing system 100 can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming console, a handheld game console, or an online game console. In one embodiment, the data processing system 100 is a mobile phone, smart phone, tablet computing device or mobile Internet device. The data processing system 100 can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, smart eyewear device, augmented reality device, or virtual reality device. In one embodiment, the data processing system 100 is a television or set top box device having one or more processors 102 and a graphical interface generated by one or more graphics processors 108.

[0027] The one or more processors 102 each include one or more processor cores 107 to process instructions which, when executed, perform operations for system and user software. In one embodiment, each of the one or more processor cores 107 is configured to process a specific instruction set 109. The instruction set 109 may facilitate complex instruction set computing (CISC), reduced instruction set computing (RISC), or computing via a very long instruction word (VLIW). Multiple processor cores 107 may each process a different instruction set 109 which may include instructions to facilitate the emulation of other instruction sets. A processor core 107 may also include other processing devices, such a digital signal processor (DSP).

[0028] In one embodiment, the processor 102 includes cache memory 104. Depending on the architecture, the processor 102 can have a single internal cache or multiple levels of internal cache. In one embodiment, the cache memory is shared among various components of the processor 102. In one embodiment, the processor 102 also uses an external cache (e.g., a Level 3 (L3) cache or last level cache (LLC)) (not shown) which may be shared among the processor cores 107 using known cache coherency techniques. A register file 106 is additionally included in the processor 102 which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). Some registers may be general-purpose registers, while other registers may be specific to the design of the processor 102.

[0029] The processor 102 is coupled to a processor bus 110 to transmit data signals between the processor 102 and other components in the system 100. The system 100 uses an exemplary 'hub' system architecture, including a memory controller hub 116 and an input output (I/O) controller hub 130. The memory controller hub 116 facilitates communication between a memory device and other components of the system 100, while the I/O controller hub (ICH) 130 provides connections to I/O devices via a local I/O bus.

[0030] The memory device 120, can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or some other memory device having suitable performance to serve as process memory. The memory 120 can store data 122 and instructions 121 for use when the processor 102 executes a process. The memory controller hub 116 also couples with an optional external graphics processor 112, which may communicate with the one or more graphics processors 108 in the processors 102 to perform graphics and media operations.

[0031] The ICH 130 enables peripherals to connect to the memory 120 and processor 102 via a high-speed I/O bus. The I/O peripherals include an audio controller 146, a firmware interface 128, a wireless transceiver 126 (e.g., Wi-Fi, Bluetooth), a data storage device 124 (e.g., hard disk drive, flash memory, etc.), and a legacy I/O controller for coupling legacy (e.g., Personal System 2 (PS/2)) devices

to the system. One or more Universal Serial Bus (USB) controllers 142 connect input devices, such as keyboard and mouse 144 combinations. A network controller 134 may also couple to the ICH 130. In one embodiment, a high-performance network controller (not shown) couples to the processor bus 110.

[0032] **Figure 7** is a block diagram of an embodiment of a processor 200 having one or more processor cores 202A-N, an integrated memory controller 214, and an integrated graphics processor 208. The processor 200 can include additional cores up to and including additional core 202N represented by the dashed lined boxes. Each of the cores 202A-N includes one or more internal cache units 204A-N. In one embodiment each core also has access to one or more shared cache units 206.

[0033] The internal cache units 204A-N and shared cache units 206 represent a cache memory hierarchy within the processor 200. The cache memory hierarchy may include at least one level of instruction and data cache within each core and one or more levels of shared mid-level cache, such as a level 2 (L2), level 3 (L3), level 4 (L4), or other levels of cache, where the highest level of cache before external memory is classified as the last level cache (LLC). In one embodiment, cache coherency logic maintains coherency between the various cache units 206 and 204A-N.

[0034] The processor 200 may also include a set of one or more bus controller units 216 and a system agent 210. The one or more bus controller units manage a set of peripheral buses, such as one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express). The system agent 210 provides management functionality for the various processor components. In one embodiment, the system agent 210 includes one or more integrated memory controllers 214 to manage access to various external memory devices (not shown).

[0035] In one embodiment, one or more of the cores 202A-N include support for simultaneous multi-threading. In such embodiment, the system agent 210 includes components for coordinating and operating cores 202A-N during multi-threaded processing. The system agent 210 may additionally include a power control unit

(PCU), which includes logic and components to regulate the power state of the cores 202A-N and the graphics processor 208.

[0036] The processor 200 additionally includes a graphics processor 208 to execute graphics processing operations. In one embodiment, the graphics processor 208 couples with the set of shared cache units 206, and the system agent unit 210, including the one or more integrated memory controllers 214. In one embodiment, a display controller 211 is coupled with the graphics processor 208 to drive graphics processor output to one or more coupled displays. The display controller 211 may be separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 208 or system agent 210.

[0037] In one embodiment a ring based interconnect unit 212 is used to couple the internal components of the processor 200, however an alternative interconnect unit may be used, such as a point to point interconnect, a switched interconnect, or other techniques, including techniques well known in the art. In one embodiment, the graphics processor 208 couples with the ring interconnect 212 via an I/O link 213.

[0038] The exemplary I/O link 213 represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded memory module 218, such as an eDRAM module. In one embodiment each of the cores 202-N and the graphics processor 208 use the embedded memory modules 218 as shared last level cache.

[0039] In one embodiment cores 202A-N are homogenous cores executing the same instruction set architecture. In another embodiment, the cores 202A-N are heterogeneous in terms of instruction set architecture (ISA), where one or more of the cores 202A-N execute a first instruction set, while at least one of the other cores executes a subset of the first instruction set or a different instruction set.

[0040] The processor 200 can be a part of or implemented on one or more substrates using any of a number of process technologies, for example, Complementary metal-oxide-semiconductor (CMOS), Bipolar

Junction/Complementary metal-oxide-semiconductor (BiCMOS) or N-type metal-oxide-semiconductor logic (NMOS). Additionally, the processor 200 can be implemented on one or more chips or as a system on a chip (SOC) integrated circuit having the illustrated components, in addition to other components.

[0041] **Figure 8** is a block diagram of one embodiment of a graphics processor 300 which may be a discrete graphics processing unit, or may be graphics processor integrated with a plurality of processing cores. In one embodiment, the graphics processor is communicated with via a memory mapped I/O interface to registers on the graphics processor and via commands placed into the processor memory. The graphics processor 300 includes a memory interface 314 to access memory. The memory interface 314 can be an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0042] The graphics processor 300 also includes a display controller 302 to drive display output data to a display device 320. The display controller 302 includes hardware for one or more overlay planes for the display and composition of multiple layers of video or user interface elements. In one embodiment the graphics processor 300 includes a video codec engine 306 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

[0043] In one embodiment, the graphics processor 300 includes a block image transfer (BLIT) engine 304 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in one embodiment, 2D graphics operations are performed using one or more components of the graphics-processing engine (GPE) 310. The graphics-processing engine 310 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0044] The GPE 310 includes a 3D pipeline 312 for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). The 3D pipeline 312 includes programmable and fixed function elements that perform various tasks within the element and/or spawn execution threads to a 3D/Media sub-system 315. While the 3D pipeline 312 can be used to perform media operations, an embodiment of the GPE 310 also includes a media pipeline 316 that is specifically used to perform media operations, such as video post processing and image enhancement.

[0045] In one embodiment, the media pipeline 316 includes fixed function or programmable logic units to perform one or more specialized media operations, such as video decode acceleration, video de-interlacing, and video encode acceleration in place of, or on behalf of the video codec engine 306. In one embodiment, the media pipeline 316 additionally includes a thread spawning unit to spawn threads for execution on the 3D/Media sub-system 315. The spawned threads perform computations for the media operations on one or more graphics execution units included in the 3D/Media sub-system.

[0046] The 3D/Media subsystem 315 includes logic for executing threads spawned by the 3D pipeline 312 and media pipeline 316. In one embodiment, the pipelines send thread execution requests to the 3D/Media subsystem 315, which includes thread dispatch logic for arbitrating and dispatching the various requests to available thread execution resources. The execution resources include an array of graphics execution units to process the 3D and media threads. In one embodiment, the 3D/Media subsystem 315 includes one or more internal caches for thread instructions and data. In one embodiment, the subsystem also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

[0047] **Figure 9** is a block diagram of an embodiment of a graphics processing engine 410 for a graphics processor. In one embodiment, the graphics processing engine (GPE) 410 is a version of the GPE 310 shown in Figure 8. The GPE 410 includes a 3D pipeline 412 and a media pipeline 416, each of which can be either

different from or similar to the implementations of the 3D pipeline 312 and the media pipeline 316 of Figure 8.

[0048] In one embodiment, the GPE 410 couples with a command streamer 403, which provides a command stream to the GPE 3D and media pipelines 412, 416. The command streamer 403 is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. The command streamer 403 receives commands from the memory and sends the commands to the 3D pipeline 412 and/or media pipeline 416. The 3D and media pipelines process the commands by performing operations via logic within the respective pipelines or by dispatching one or more execution threads to the execution unit array 414. In one embodiment, the execution unit array 414 is scalable, such that the array includes a variable number of execution units based on the target power and performance level of the GPE 410.

[0049] A sampling engine 430 couples with memory (e.g., cache memory or system memory) and the execution unit array 414. In one embodiment, the sampling engine 430 provides a memory access mechanism for the scalable execution unit array 414 that allows the execution array 414 to read graphics and media data from memory. In one embodiment, the sampling engine 430 includes logic to perform specialized image sampling operations for media.

[0050] The specialized media sampling logic in the sampling engine 430 includes a de-noise/de-interlace module 432, a motion estimation module 434, and an image scaling and filtering module 436. The de-noise/de-interlace module 432 includes logic to perform one or more of a de-noise or a de-interlace algorithm on decoded video data. The de-interlace logic combines alternating fields of interlaced video content into a single frame of video. The de-noise logic reduces or remove data noise from video and image data. In one embodiment, the de-noise logic and de-interlace logic are motion adaptive and use spatial or temporal filtering based on the amount of motion detected in the video data. In one embodiment, the de-noise / de-interlace module 432 includes dedicated motion detection logic (e.g., within the motion estimation engine 434).

[0051] The motion estimation engine 434 provides hardware acceleration for video operations by performing video acceleration functions such as motion vector estimation and prediction on video data. The motion estimation engine determines motion vectors that describe the transformation of image data between successive video frames. In one embodiment, a graphics processor media codec uses the video motion estimation engine 434 to perform operations on video at the macro-block level that may otherwise be computationally intensive to perform using a general-purpose processor. In one embodiment, the motion estimation engine 434 is generally available to graphics processor components to assist with video decode and processing functions that are sensitive or adaptive to the direction or magnitude of the motion within video data.

[0052] The image scaling and filtering module 436 performs image-processing operations to enhance the visual quality of generated images and video. In one embodiment, the scaling and filtering module 436 processes image and video data during the sampling operation before providing the data to the execution unit array 414.

[0053] In one embodiment, the graphics processing engine 410 includes a data port 444, which provides an additional mechanism for graphics subsystems to access memory. The data port 444 facilitates memory access for operations including render target writes, constant buffer reads, scratch memory space reads/writes, and media surface accesses. In one embodiment, the data port 444 includes cache memory space to cache accesses to memory. The cache memory can be a single data cache or separated into multiple caches for the multiple subsystems that access memory via the data port (e.g., a render buffer cache, a constant buffer cache, etc.). In one embodiment, threads executing on an execution unit in the execution unit array 414 communicate with the data port by exchanging messages via a data distribution interconnect that couples each of the sub-systems of the graphics processing engine 410.

[0054] **Figure 10** is a block diagram of another embodiment of a graphics processor. In one embodiment, the graphics processor includes a ring interconnect 502, a pipeline front-end 504, a media engine 537, and graphics cores 580A-N. The

ring interconnect 502 couples the graphics processor to other processing units, including other graphics processors or one or more general-purpose processor cores. In one embodiment, the graphics processor is one of many processors integrated within a multi-core processing system.

[0055] The graphics processor receives batches of commands via the ring interconnect 502. The incoming commands are interpreted by a command streamer 503 in the pipeline front-end 504. The graphics processor includes scalable execution logic to perform 3D geometry processing and media processing via the graphics core(s) 580A-N. For 3D geometry processing commands, the command streamer 503 supplies the commands to the geometry pipeline 536. For at least some media processing commands, the command streamer 503 supplies the commands to a video front end 534, which couples with a media engine 537. The media engine 537 includes a video quality engine (VQE) 530 for video and image post processing and a multi-format encode/decode (MFX) 533 engine to provide hardware-accelerated media data encode and decode. The geometry pipeline 536 and media engine 537 each generate execution threads for the thread execution resources provided by at least one graphics core 580A.

[0056] The graphics processor includes scalable thread execution resources featuring modular cores 580A-N (sometime referred to as core slices), each having multiple sub-cores 550A-N, 560A-N (sometimes referred to as core sub-slices). The graphics processor can have any number of graphics cores 580A through 580N. In one embodiment, the graphics processor includes a graphics core 580A having at least a first sub-core 550A and a second core sub-core 560A. In another embodiment, the graphics processor is a low power processor with a single sub-core (e.g., 550A). In one embodiment, the graphics processor includes multiple graphics cores 580A-N, each including a set of first sub-cores 550A-N and a set of second sub-cores 560A-N. Each sub-core in the set of first sub-cores 550A-N includes at least a first set of execution units 552A-N and media/texture samplers 554A-N. Each sub-core in the set of second sub-cores 560A-N includes at least a second set of execution units 562A-N and samplers 564A-N. In one embodiment, each sub-core 550A-N, 560A-N shares a set of shared resources 570A-N. In one embodiment, the

shared resources include shared cache memory and pixel operation logic. Other shared resources may also be included in the various embodiments of the graphics processor.

[0057] **Figure 11** illustrates thread execution logic 600 including an array of processing elements employed in one embodiment of a graphics processing engine. In one embodiment, the thread execution logic 600 includes a pixel shader 602, a thread dispatcher 604, instruction cache 606, a scalable execution unit array including a plurality of execution units 608A-N, a sampler 610, a data cache 612, and a data port 614. In one embodiment the included components are interconnected via an interconnect fabric that links to each of the components. The thread execution logic 600 includes one or more connections to memory, such as system memory or cache memory, through one or more of the instruction cache 606, the data port 614, the sampler 610, and the execution unit array 608A-N. In one embodiment, each execution unit (e.g. 608A) is an individual vector processor capable of executing multiple simultaneous threads and processing multiple data elements in parallel for each thread. The execution unit array 608A-N includes any number individual execution units.

[0058] In one embodiment, the execution unit array 608A-N is primarily used to execute “shader” programs. In one embodiment, the execution units in the array 608A-N execute an instruction set that includes native support for many standard 3D graphics shader instructions, such that shader programs from graphics libraries (e.g., Direct 3D and OpenGL) are executed with a minimal translation. The execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, vertex shaders), pixel processing (e.g., pixel shaders, fragment shaders) and general-purpose processing (e.g., compute and media shaders).

[0059] Each execution unit in the execution unit array 608A-N operates on arrays of data elements. The number of data elements is the “execution size,” or the number of channels for the instruction. An execution channel is a logical unit of execution for data element access, masking, and flow control within instructions. The number of channels may be independent of the number of physical ALUs or FPUs for a

particular graphics processor. The execution units 608A-N support integer and floating-point data types.

[0060] The execution unit instruction set includes single instruction multiple data (SIMD) instructions. The various data elements can be stored as a packed data type in a register and the execution unit will process the various elements based on the data size of the elements. For example, when operating on a 256-bit wide vector, the 256 bits of the vector are stored in a register and the execution unit operates on the vector as four separate 64-bit packed data elements (quad-word (QW) size data elements), eight separate 32-bit packed data elements (double word (DW) size data elements), sixteen separate 16-bit packed data elements (word (W) size data elements), or thirty-two separate 8-bit data elements (byte (B) size data elements). However, different vector widths and register sizes are possible.

[0061] One or more internal instruction caches (e.g., 606) are included in the thread execution logic 600 to cache thread instructions for the execution units. In one embodiment, one or more data caches (e.g., 612) are included to cache thread data during thread execution. A sampler 610 is included to provide texture sampling for 3D operations and media sampling for media operations. In one embodiment, the sampler 610 includes specialized texture or media sampling functionality to process texture or media data during the sampling process before providing the sampled data to an execution unit.

[0062] During execution, the graphics and media pipelines send thread initiation requests to the thread execution logic 600 via thread spawning and dispatch logic. The thread execution logic 600 includes a local thread dispatcher 604 that arbitrates thread initiation requests from the graphics and media pipelines and instantiates the requested threads on one or more execution units 608A-N. For example, the geometry pipeline (e.g., 536 of Figure 6) dispatches vertex processing, tessellation, or geometry processing threads to the thread execution logic 600. The thread dispatcher 604 can also process runtime thread spawning requests from the executing shader programs.

[0063] Once a group of geometric objects have been processed and rasterized into pixel data, the pixel shader 602 is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In one embodiment, the pixel shader 602 calculates the values of the various vertex attributes that are to be interpolated across the rasterized object. The pixel shader 602 then executes an API-supplied pixel shader program. To execute the pixel shader program, the pixel shader 602 dispatches threads to an execution unit (e.g., 608A) via the thread dispatcher 604. The pixel shader 602 uses texture sampling logic in the sampler 610 to access texture data in texture maps stored in memory. Arithmetic operations on the texture data and the input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

[0064] In one embodiment, the data port 614 provides a memory access mechanism for the thread execution logic 600 output processed data to memory for processing on a graphics processor output pipeline. In one embodiment, the data port 614 includes or couples to one or more cache memories (e.g., data cache 612) to cache data for memory access via the data port.

[0065] **Figure 12** is a block diagram illustrating a graphics processor execution unit instruction format according to an embodiment. In one embodiment, the graphics processor execution units support an instruction set having instructions in multiple formats. The solid lined boxes illustrate the components that are generally included in an execution unit instruction, while the dashed lines include components that are optional or that are only included in a sub-set of the instructions. The instruction format described and illustrated are macro-instructions, in that they are instructions supplied to the execution unit, as opposed to micro-operations resulting from instruction decode once the instruction is processed.

[0066] In one embodiment, the graphics processor execution units natively support instructions in a 128-bit format 710. A 64-bit compacted instruction format 730 is available for some instructions based on the selected instruction, instruction options, and number of operands. The native 128-bit format 710 provides access to all instruction options, while some options and operations are restricted in the 64-bit

format 730. The native instructions available in the 64-bit format 730 varies by embodiment. In one embodiment, the instruction is compacted in part using a set of index values in an index field 713. The execution unit hardware references a set of compaction tables based on the index values and uses the compaction table outputs to reconstruct a native instruction in the 128-bit format 710.

[0067] For each format, an instruction opcode 712 defines the operation that the execution unit is to perform. The execution units execute each instruction in parallel across the multiple data elements of each operand. For example, in response to an add instruction the execution unit performs a simultaneous add operation across each color channel representing a texture element or picture element. By default, the execution unit performs each instruction across all data channels of the operands. An instruction control field 712 enables control over certain execution options, such as channels selection (e.g., predication) and data channel order (e.g., swizzle). For 128-bit instructions 710 an exec-size field 716 limits the number of data channels that will be executed in parallel. The exec-size field 716 is not available for use in the 64-bit compact instruction format 730.

[0068] Some execution unit instructions have up to three operands including two source operands, src0 720, src1 722, and one destination 718. In one embodiment, the execution units support dual destination instructions, where one of the destinations is implied. Data manipulation instructions can have a third source operand (e.g., SRC2 724), where the instruction opcode JJ12 determines the number of source operands. An instruction's last source operand can be an immediate (e.g., hard-coded) value passed with the instruction.

[0069] In one embodiment instructions are grouped based on opcode bit-fields to simplify Opcode decode 740. For an 8-bit opcode, bits 4, 5, and 6 allow the execution unit to determine the type of opcode. The precise opcode grouping shown is exemplary. In one embodiment, a move and logic opcode group 742 includes data movement and logic instructions (e.g., mov, cmp). The move and logic group 742 shares the five most significant bits (MSB), where move instructions are in the form of 0000xxxxb (e.g., 0x0x) and logic instructions are in the form of 0001xxxxb (e.g., 0x01). A flow control instruction group 744 (e.g., call, jmp) includes

instructions in the form of 0010xxxxb (e.g., 0x20). A miscellaneous instruction group 746 includes a mix of instructions, including synchronization instructions (e.g., wait, send) in the form of 0011xxxxb (e.g., 0x30). A parallel math instruction group 748 includes component-wise arithmetic instructions (e.g., add, mul) in the form of 0100xxxxb (e.g., 0x40). The parallel math group 748 performs the arithmetic operations in parallel across data channels. The vector math group 750 includes arithmetic instructions (e.g., dp4) in the form of 0101xxxxb (e.g., 0x50). The vector math group performs arithmetic such as dot product calculations on vector operands.

[0070] **Figure 13** is a block diagram of another embodiment of a graphics processor which includes a graphics pipeline 820, a media pipeline 830, a display engine 840, thread execution logic 850, and a render output pipeline 870. In one embodiment, the graphics processor is a graphics processor within a multi-core processing system that includes one or more general purpose processing cores. The graphics processor is controlled by register writes to one or more control registers (not shown) or via commands issued to the graphics processor via a ring interconnect 802. The ring interconnect 802 couples the graphics processor to other processing components, such as other graphics processors or general-purpose processors. Commands from the ring interconnect are interpreted by a command streamer 803 which supplies instructions to individual components of the graphics pipeline 820 or media pipeline 830.

[0071] The command streamer 803 directs the operation of a vertex fetcher 805 component that reads vertex data from memory and executes vertex-processing commands provided by the command streamer 803. The vertex fetcher 805 provides vertex data to a vertex shader 807, which performs coordinate space transformation and lighting operations to each vertex. The vertex fetcher 805 and vertex shader 807 execute vertex-processing instructions by dispatching execution threads to the execution units 852A, 852B via a thread dispatcher 831.

[0072] In one embodiment, the execution units 852A, 852B are an array of vector processors having an instruction set for performing graphics and media operations. The execution units 852A, 852B have an attached L1 cache 851 that is specific for each array or shared between the arrays. The cache can be configured as a data

cache, an instruction cache, or a single cache that is partitioned to contain data and instructions in different partitions.

[0073] In one embodiment, the graphics pipeline 820 includes tessellation components to perform hardware-accelerated tessellation of 3D objects. A programmable hull shader 811 configures the tessellation operations. A programmable domain shader 817 provides back-end evaluation of tessellation output. A tessellator 813 operates at the direction of the hull shader 811 and contains special purpose logic to generate a set of detailed geometric objects based on a coarse geometric model that is provided as input to the graphics pipeline 820. If tessellation is not used, the tessellation components 811, 813, 817 can be bypassed.

[0074] The complete geometric objects can be processed by a geometry shader 819 via one or more threads dispatched to the execution units 852A, 852B, or can proceed directly to the clipper 829. The geometry shader operates on entire geometric objects, rather than vertices or patches of vertices as in previous stages of the graphics pipeline. If the tessellation is disabled the geometry shader 819 receives input from the vertex shader 807. The geometry shader 819 is programmable by a geometry shader program to perform geometry tessellation if the tessellation units are disabled.

[0075] Prior to rasterization, vertex data is processed by a clipper 829, which is either a fixed function clipper or a programmable clipper having clipping and geometry shader functions. In one embodiment, a rasterizer 873 in the render output pipeline 870 dispatches pixel shaders to convert the geometric objects into their per pixel representations. In one embodiment, pixel shader logic is included in the thread execution logic 850.

[0076] The graphics engine has an interconnect bus, interconnect fabric, or some other interconnect mechanism that allows data and message passing amongst the major components of the graphics engine. In one embodiment the execution units 852A, 852B and associated cache(s) 851, texture and media sampler 854, and texture/sampler cache 858 interconnect via a data port 856 to perform memory

access and communicate with render output pipeline components of the graphics engine. In one embodiment, the sampler 854, caches 851, 858 and execution units 852A, 852B each have separate memory access paths.

[0077] In one embodiment, the render output pipeline 870 contains a rasterizer and depth test component 873 that converts vertex-based objects into their associated pixel-based representation. In one embodiment, the rasterizer logic includes a windower/masker unit to perform fixed function triangle and line rasterization. An associated render and depth buffer caches 878, 879 are also available in one embodiment. A pixel operations component 877 performs pixel-based operations on the data, though in some instances, pixel operations associated with 2D operations (e.g. bit block image transfers with blending) are performed by the 2D engine 841, or substituted at display time by the display controller 843 using overlay display planes. In one embodiment a shared L3 cache 875 is available to all graphics components, allowing the sharing of data without the use of main system memory.

[0078] The graphics processor media pipeline 830 includes a media engine 837 and a video front end 834. In one embodiment, the video front end 834 receives pipeline commands from the command streamer 803. However, in one embodiment the media pipeline 830 includes a separate command streamer. The video front-end 834 processes media commands before sending the command to the media engine 837. In one embodiment, the media engine includes thread spawning functionality to spawn threads for dispatch to the thread execution logic 850 via the thread dispatcher 831.

[0079] In one embodiment, the graphics engine includes a display engine 840. In one embodiment, the display engine 840 is external to the graphics processor and couples with the graphics processor via the ring interconnect 802, or some other interconnect bus or fabric. The display engine 840 includes a 2D engine 841 and a display controller 843. The display engine 840 contains special purpose logic capable of operating independently of the 3D pipeline. The display controller 843 couples with a display device (not shown), which may be a system integrated display device, as in a laptop computer, or an external display device attached via an display device connector.

[0080] The graphics pipeline 820 and media pipeline 830 are configurable to perform operations based on multiple graphics and media programming interfaces and are not specific to any one application programming interface (API). In one embodiment, driver software for the graphics processor translates API calls that are specific to a particular graphics or media library into commands that can be processed by the graphics processor. In various embodiments, support is provided for the Open Graphics Library (OpenGL) and Open Computing Language (OpenCL) supported by the Khronos Group, the Direct3D library from the Microsoft Corporation, or, in one embodiment, both OpenGL and D3D. Support may also be provided for the Open Source Computer Vision Library (OpenCV). A future API with a compatible 3D pipeline would also be supported if a mapping can be made from the pipeline of the future API to the pipeline of the graphics processor.

[0081] **Figure 14A** is a block diagram illustrating a graphics processor command format according to an embodiment and **Figure 14B** is a block diagram illustrating a graphics processor command sequence according to an embodiment. The solid lined boxes in Figure 14A illustrate the components that are generally included in a graphics command while the dashed lines include components that are optional or that are only included in a sub-set of the graphics commands. The exemplary graphics processor command format 900 of Figure 14A includes data fields to identify a target client 902 of the command, a command operation code (opcode) 904, and the relevant data 906 for the command. A sub-opcode 905 and a command size 908 are also included in some commands.

[0082] The client 902 specifies the client unit of the graphics device that processes the command data. In one embodiment, a graphics processor command parser examines the client field of each command to condition the further processing of the command and route the command data to the appropriate client unit. In one embodiment, the graphics processor client units include a memory interface unit, a render unit, a 2D unit, a 3D unit, and a media unit. Each client unit has a corresponding processing pipeline that processes the commands. Once the command is received by the client unit, the client unit reads the opcode 904 and, if present, sub-opcode 905 to determine the operation to perform. The client unit

performs the command using information in the data 906 field of the command. For some commands an explicit command size 908 is expected to specify the size of the command. In one embodiment, the command parser automatically determines the size of at least some of the commands based on the command opcode. In one embodiment commands are aligned via multiples of a double word.

[0083] The flow chart in Figure 14B shows a sample command sequence 910. In one embodiment, software or firmware of a data processing system that features an embodiment of the graphics processor uses a version of the command sequence shown to set up, execute, and terminate a set of graphics operations. A sample command sequence is shown and described for exemplary purposes, however embodiments are not limited to these commands or to this command sequence. Moreover, the commands may be issued as batch of commands in a command sequence, such that the graphics processor will process the sequence of commands in an at least partially concurrent manner.

[0084] The sample command sequence 910 may begin with a pipeline flush command 912 to cause any active graphics pipeline to complete the currently pending commands for the pipeline. In one embodiment, the 3D pipeline 922 and the media pipeline 924 do not operate concurrently. The pipeline flush is performed to cause the active graphics pipeline to complete any pending commands. In response to a pipeline flush, the command parser for the graphics processor will pause command processing until the active drawing engines complete pending operations and the relevant read caches are invalidated. Optionally, any data in the render cache that is marked 'dirty' can be flushed to memory. A pipeline flush command 912 can be used for pipeline synchronization or before placing the graphics processor into a low power state.

[0085] A pipeline select command 913 is used when a command sequence requires the graphics processor to explicitly switch between pipelines. A pipeline select command 913 is required only once within an execution context before issuing pipeline commands unless the context is to issue commands for both pipelines. In one embodiment, a pipeline flush command is 912 is required immediately before a pipeline switch via the pipeline select command 913.

[0086] A pipeline control command 914 configures a graphics pipeline for operation and is used to program the 3D pipeline 922 and the media pipeline 924. The pipeline control command 914 configures the pipeline state for the active pipeline. In one embodiment, the pipeline control command 914 is used for pipeline synchronization and to clear data from one or more cache memories within the active pipeline before processing a batch of commands.

[0087] Return buffer state commands 916 are used to configure a set of return buffers for the respective pipelines to write data. Some pipeline operations require the allocation, selection, or configuration of one or more return buffers into which the operations write intermediate data during processing. The graphics processor also uses one or more return buffers to store output data and to perform cross thread communication. The return buffer state 916 includes selecting the size and number of return buffers to use for a set of pipeline operations.

[0088] The remaining commands in the command sequence differ based on the active pipeline for operations. Based on a pipeline determination 920, the command sequence is tailored to the 3D pipeline 922 beginning with the 3D pipeline state 930, or the media pipeline 924 beginning at the media pipeline state 940.

[0089] The commands for the 3D pipeline state 930 include 3D state setting commands for vertex buffer state, vertex element state, constant color state, depth buffer state, and other state variables that are to be configured before 3D primitive commands are processed. The values of these commands are determined at least in part based the particular 3D API in use. 3D pipeline state 930 commands are also able to selectively disable or bypass certain pipeline elements if those elements will not be used.

[0090] The 3D primitive 932 command is used to submit 3D primitives to be processed by the 3D pipeline. Commands and associated parameters that are passed to the graphics processor via the 3D primitive 932 command are forwarded to the vertex fetch function in the graphics pipeline. The vertex fetch function uses the 3D primitive 932 command data to generate vertex data structures. The vertex data structures are stored in one or more return buffers. The 3D primitive 932

command is used to perform vertex operations on 3D primitives via vertex shaders. To process vertex shaders, the 3D pipeline 922 dispatches shader execution threads to graphics processor execution units.

[0091] The 3D pipeline 922 is triggered via an execute 934 command or event. In one embodiment a register write triggers command execution. In one embodiment execution is triggered via a 'go' or 'kick' command in the command sequence. In one embodiment command execution is triggered using a pipeline synchronization command to flush the command sequence through the graphics pipeline. The 3D pipeline will perform geometry processing for the 3D primitives. Once operations are complete, the resulting geometric objects are rasterized and the pixel engine colors the resulting pixels. Additional commands to control pixel shading and pixel back end operations may also be included for those operations.

[0092] The sample command sequence 910 follows the media pipeline 924 path when performing media operations. In general, the specific use and manner of programming for the media pipeline 924 depends on the media or compute operations to be performed. Specific media decode operations may be offloaded to the media pipeline during media decode. The media pipeline can also be bypassed and media decode can be performed in whole or in part using resources provided by one or more general purpose processing cores. In one embodiment, the media pipeline also includes elements for general-purpose graphics processor unit (GPGPU) operations, where the graphics processor is used to perform SIMD vector operations using computational shader programs that are not explicitly related to the rendering of graphics primitives.

[0093] The media pipeline 924 is configured in a similar manner as the 3D pipeline 922. A set of media pipeline state commands 940 are dispatched or placed into a command queue before the media object commands 942. The media pipeline state commands 940 include data to configure the media pipeline elements that will be used to process the media objects. This includes data to configure the video decode and video encode logic within the media pipeline, such as encode or decode format. The media pipeline state commands 940 also support the use one or more pointers to "indirect" state elements that contain a batch of state settings.

[0094] Media object commands 942 supply pointers to media objects for processing by the media pipeline. The media objects include memory buffers containing video data to be processed. In one embodiment, all media pipeline state must be valid before issuing a media object command 942. Once the pipeline state is configured and media object commands 942 are queued, the media pipeline 924 is triggered via an execute 934 command or an equivalent execute event (e.g., register write). Output from the media pipeline 924 may then be post processed by operations provided by the 3D pipeline 922 or the media pipeline 924. In one embodiment, GPGPU operations are configured and executed in a similar manner as media operations.

[0095] **Figure 15** illustrates exemplary graphics software architecture for a data processing system according to an embodiment. The software architecture includes a 3D graphics application 1010, an operating system 1020, and at least one processor 1030. The processor 1030 includes a graphics processor 1032 and one or more general-purpose processor core(s) 1034. The graphics application 1010 and operating system 1020 each execute in the system memory 1050 of the data processing system.

[0096] In one embodiment, the 3D graphics application 1010 contains one or more shader programs including shader instructions 1012. The shader language instructions may be in a high-level shader language, such as the High Level Shader Language (HLSL) or the OpenGL Shader Language (GLSL). The application also includes executable instructions 1014 in a machine language suitable for execution by the general-purpose processor core 1034. The application also includes graphics objects 1016 defined by vertex data.

[0097] The operating system 1020 may be a Microsoft® Windows® operating system from the Microsoft Corporation, a proprietary UNIX-like operating system, or an open source UNIX-like operating system using a variant of the Linux kernel. When the Direct3D API is in use, the operating system 1020 uses a front-end shader compiler 1024 to compile any shader instructions 1012 in HLSL into a lower-level shader language. The compilation may be a just-in-time compilation or the application can perform share pre-compilation. In one embodiment, high-level

shaders are compiled into low-level shaders during the compilation of the 3D graphics application 1010.

[0098] The user mode graphics driver 1026 may contain a back-end shader compiler 1027 to convert the shader instructions 1012 into a hardware specific representation. When the OpenGL API is in use, shader instructions 1012 in the GLSL high-level language are passed to a user mode graphics driver 1026 for compilation. The user mode graphics driver uses operating system kernel mode functions 1028 to communicate with a kernel mode graphics driver 1029. The kernel mode graphics driver 1029 communicates with the graphics processor 1032 to dispatch commands and instructions.

[0099] To the extent various operations or functions are described herein, they can be described or defined as hardware circuitry, software code, instructions, configuration, and/or data. The content can be embodied in hardware logic, or as directly executable software ("object" or "executable" form), source code, high level shader code designed for execution on a graphics engine, or low level assembly language code in an instruction set for a specific processor or graphics core. The software content of the embodiments described herein can be provided via an article of manufacture with the content stored thereon, or via a method of operating a communication interface to send data via the communication interface.

[0100] A non-transitory machine readable storage medium can cause a machine to perform the functions or operations described, and includes any mechanism that stores information in a form accessible by a machine (e.g., computing device, electronic system, etc.), such as recordable/non-recordable media (e.g., read only memory (ROM), random access memory (RAM), magnetic disk storage media, optical storage media, flash memory devices, etc.). A communication interface includes any mechanism that interfaces to any of a hardwired, wireless, optical, etc., medium to communicate to another device, such as a memory bus interface, a processor bus interface, an Internet connection, a disk controller, etc. The communication interface is configured by providing configuration parameters or sending signals to prepare the communication interface to provide a data signal

describing the software content. The communication interface can be accessed via one or more commands or signals sent to the communication interface.

[0101] Various components described can be a means for performing the operations or functions described. Each component described herein includes software, hardware, or a combination of these. The components can be implemented as software modules, hardware modules, special-purpose hardware (e.g., application specific hardware, application specific integrated circuits (ASICs), digital signal processors (DSPs), etc.), embedded controllers, hardwired circuitry, etc. Besides what is described herein, various modifications can be made to the disclosed embodiments and implementations of the invention without departing from their scope. Therefore, the illustrations and examples herein should be construed in an illustrative, and not a restrictive sense. The scope of the invention should be measured solely by reference to the claims that follow.

[0102] The following clauses and/or examples pertain to further embodiments;

One example embodiment may be a method comprising converting from a first color space to a second color space, using a two-dimensional lookup table in said second color space and converting from said second color space to said first color space. The method may also include using more than one two-dimensional lookup table. The method may also include wherein said first color space is RGB and said second color space is YCbCr. The method of claim 1 may also include using a color pixel with N color components and processing with n number of M dimensional LUT where $M < N$ and n is a positive integer number. The method of claim 1 include using the second color space to implement a correction that only has two changing color components. The method may also include switching from the first color space wherein all three components change. The method may also include determining, for a given pixel, which of at least two two-dimensional lookup tables to use. The method may also include determining whether a pixel color is darker or lighter than a threshold. The method may also include detecting a hue of an RGB pixel, receiving hue and saturation adjustment factors from a user, converting to YCbCr color space, adjusting the Cb and Cr color components based on said factors and then converting back to RGB color space. The method may also

include converting from RGB to HSV color space, dividing a hue region into N sub-regions and using N two-dimensional lookup tables to modify S and V components.

[0103] Another example embodiment may be at least one or more non-transitory computer readable media storing instructions executed to perform a sequence comprising converting from a first color space to a second color space, using a two-dimensional lookup table in said second color space, and converting from said second color space to said first color space. The media may include said sequence including using more than one two-dimensional lookup table. The media may include said sequence wherein said first color space is RGB. The media may include said sequence wherein said second color space is YCbCr. The media may include said sequence including using the second color space to implement a correction that only has two changing color components. The media may include said sequence including switching from the first color space wherein all three components change. The media may include said sequence including determining, for a given pixel, which of at least two two-dimensional lookup tables to use. The media may include said sequence including determining whether a pixel color is darker or lighter than a threshold. The media may include said sequence including detecting a hue of an RGB pixel, receiving hue and saturation adjustment factors from a user, converting to YCbCr color space, adjusting the Cb and Cr color components based on said factors and then converting back to RGB color space. The media may include said sequence including converting from RGB to HSV color space, dividing a hue region into N sub-regions and using N two-dimensional lookup tables to modify S and V components.

[0104] In another example embodiment may be an apparatus comprising a hardware device to convert from a first color space to a second color space, use a two-dimensional lookup table in said second color space, and convert from said second color space to said first color space, and a storage coupled to said device. The apparatus may include said device to use more than one two-dimensional lookup table. The apparatus may include wherein said first color space is RGB. The apparatus may include wherein said second color space is YCbCr. The apparatus may include said device to use the second color space to implement a correction

that only has two changing color components. The apparatus may include said device to switch from the first color space wherein all three components change. The apparatus may include said device to determine, for a given pixel, which of at least two two-dimensional lookup tables to use. The apparatus may include said device to determine whether a pixel color is darker or lighter than a threshold. The apparatus may include said device to detect a hue of an RGB pixel, receiving hue and saturation adjustment factors from a user, converting to YCbCr color space, adjusting the Cb and Cr color components based on said factors and then converting back to RGB color space. The apparatus may include said device to convert from RGB to HSV color space, dividing a hue region into N sub-regions and using N two-dimensional lookup tables to modify S and V components.

[0105] The graphics processing techniques described herein may be implemented in various hardware architectures. For example, graphics functionality may be integrated within a chipset. Alternatively, a discrete graphics processor may be used. As still another embodiment, the graphics functions may be implemented by a general purpose processor, including a multicore processor.

[0106] References throughout this specification to “one embodiment” or “an embodiment” mean that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one implementation encompassed within the present disclosure. Thus, appearances of the phrase “one embodiment” or “in an embodiment” are not necessarily referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be instituted in other suitable forms other than the particular embodiment illustrated and all such forms may be encompassed within the claims of the present application.

[0107] While a limited number of embodiments have been described, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this disclosure.

What is claimed is:

- 1 1. A method comprising:
2 converting from a first color space to a second color space;
3 using a two-dimensional lookup table in said second color space; and
4 converting from said second color space to said first color space.
- 1 2. The method of claim 1 including using more than one two-dimensional lookup
2 table.
- 1 3. The method of claim 1 wherein said first color space is RGB and said second
2 color space is YCbCr.
- 1 4. The method of claim 1 including using a color pixel with N color components
2 and processing with n number of M dimensional LUT where $M < N$ and n is a positive
3 integer number.
- 1 5. The method of claim 1 including using the second color space to implement a
2 correction that only has two changing color components.
- 1 6. The method of claim 5 including switching from the first color space wherein
2 all three components change.
- 1 7. The method of claim 1 including determining, for a given pixel, which of at
2 least two two-dimensional lookup tables to use.
- 1 8. The method of claim 7 including determining whether a pixel color is darker or
2 lighter than a threshold.
- 1 9. The method of claim 1 including detecting a hue of an RGB pixel, receiving
2 hue and saturation adjustment factors from a user, converting to YCbCr color space,

3 adjusting the Cb and Cr color components based on said factors and then converting
4 back to RGB color space.

1 10. The method of claim 1 including converting from RGB to HSV color space,
2 dividing a hue region into N sub-regions and using N two-dimensional lookup tables
3 to modify S and V components.

1 11. One or more non-transitory computer readable media storing instructions
2 executed to perform a sequence comprising:
3 converting from a first color space to a second color space;
4 using a two-dimensional lookup table in said second color space; and
5 converting from said second color space to said first color space.

1 12. The media of claim 11, said sequence including using more than one two-
2 dimensional lookup table.

1 13. The media of claim 11, said sequence wherein said first color space is RGB.

1 14. The media of claim 13 wherein said second color space is YCbCr.

1 15. The media of claim 11, said sequence including using the second color space
2 to implement a correction that only has two changing color components.

1 16. The media of claim 15, said sequence including switching from the first color
2 space wherein all three components change.

1 17. The media of claim 11, said sequence including determining, for a given pixel,
2 which of at least two two-dimensional lookup tables to use.

1 18. The media of claim 17, said sequence including determining whether a pixel
2 color is darker or lighter than a threshold.

1 19. The media of claim 11, said sequence including detecting a hue of an RGB
2 pixel, receiving hue and saturation adjustment factors from a user, converting to
3 YCbCr color space, adjusting the Cb and Cr color components based on said factors
4 and then converting back to RGB color space.

1 20. The media of claim 11, said sequence including converting from RGB to HSV
2 color space, dividing a hue region into N sub-regions and using N two-dimensional
3 lookup tables to modify S and V components.

1 21. An apparatus comprising:
2 a hardware device to convert from a first color space to a second color space,
3 use a two-dimensional lookup table in said second color space, and convert from
4 said second color space to said first color space; and
5 a storage coupled to said device.

1 22. The apparatus of claim 21, said device to use more than one two-dimensional
2 lookup table.

1 23. The apparatus of claim 21 wherein said first color space is RGB.

1 24. The apparatus of claim 23 wherein said second color space is YCbCr.

1 25. The apparatus of claim 21, said device to use the second color space to
2 implement a correction that only has two changing color components.

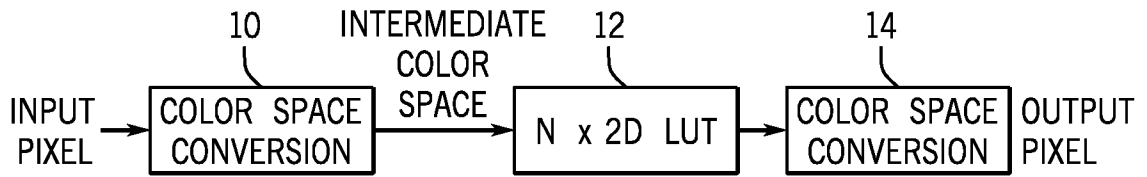


FIG. 1

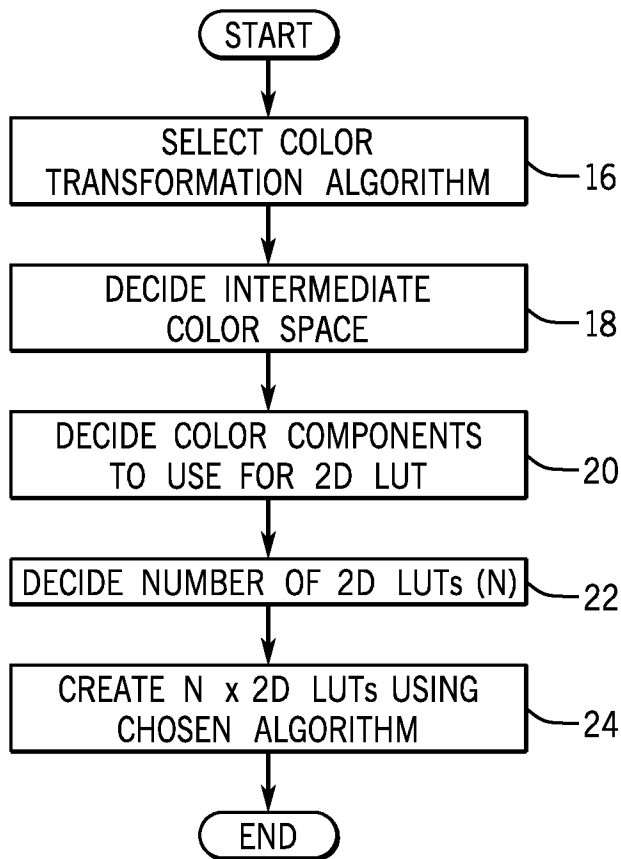


FIG. 2

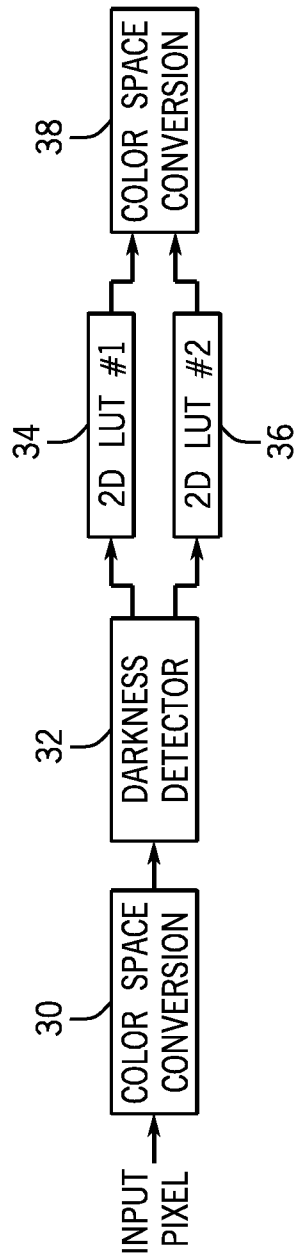


FIG. 3

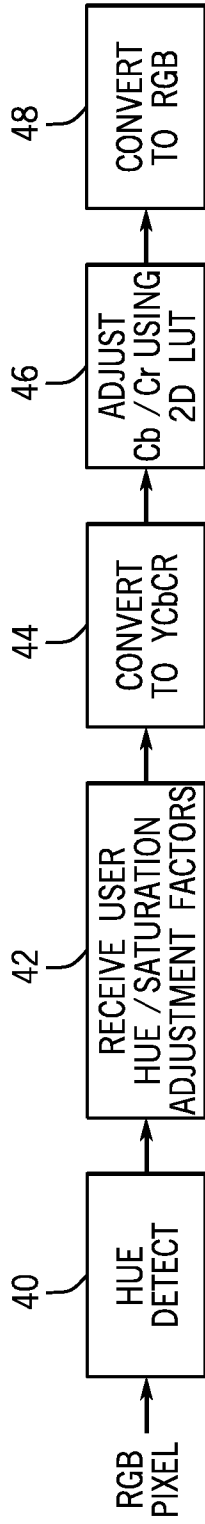


FIG. 4

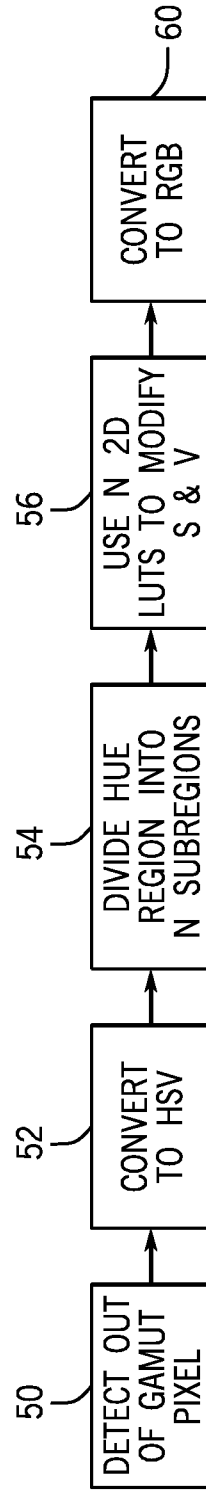


FIG. 5

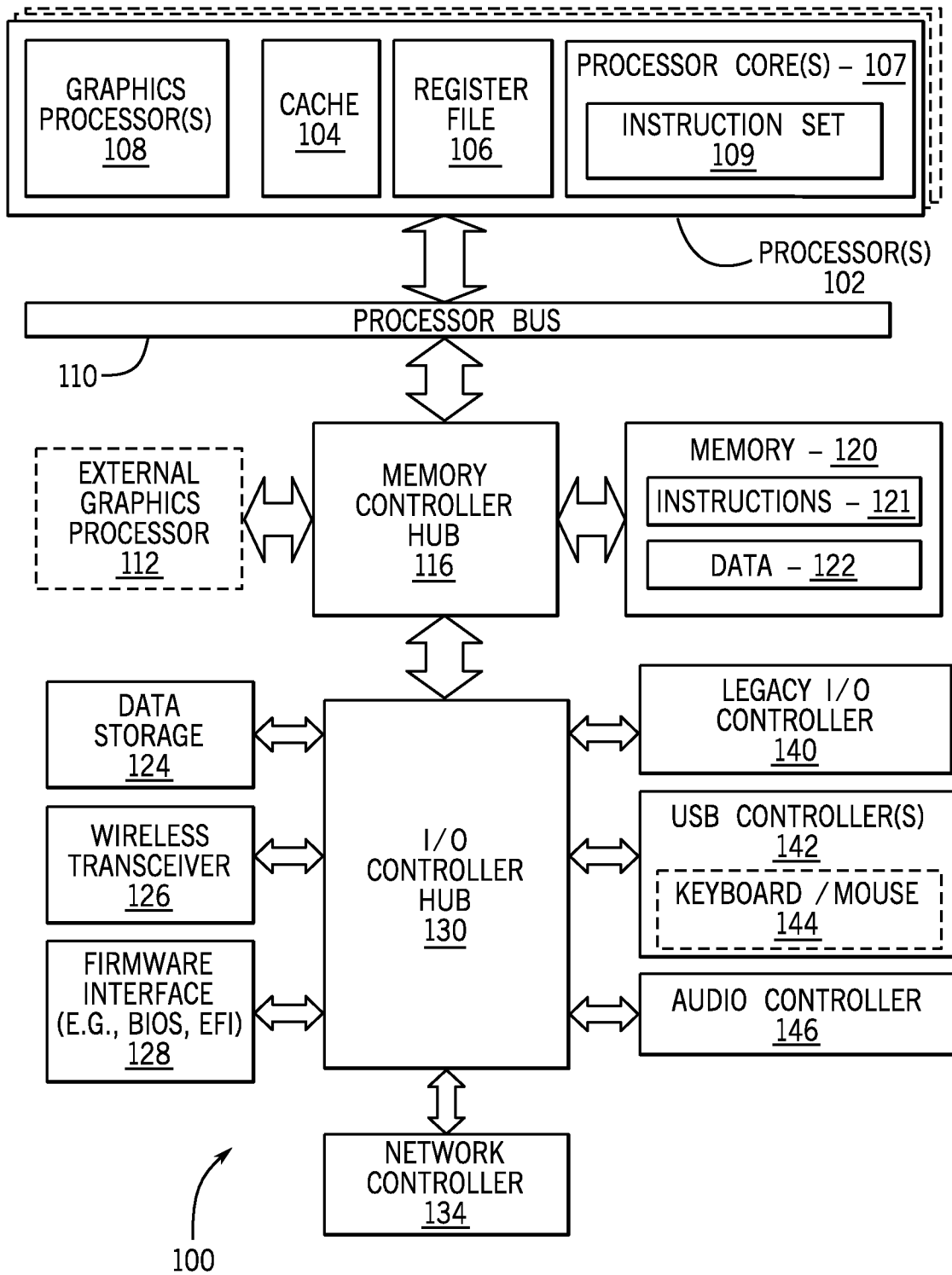


FIG. 6

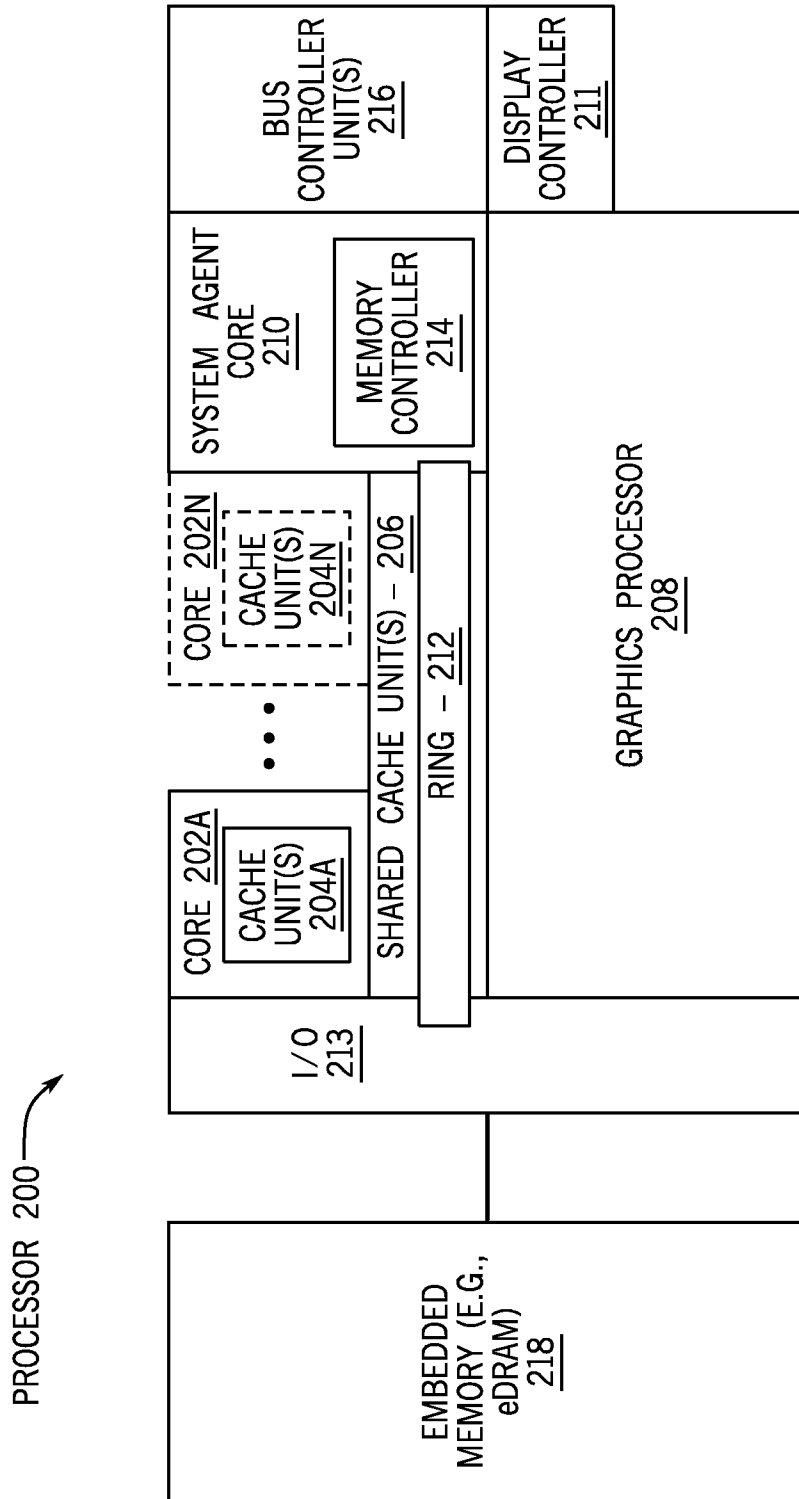


FIG. 7

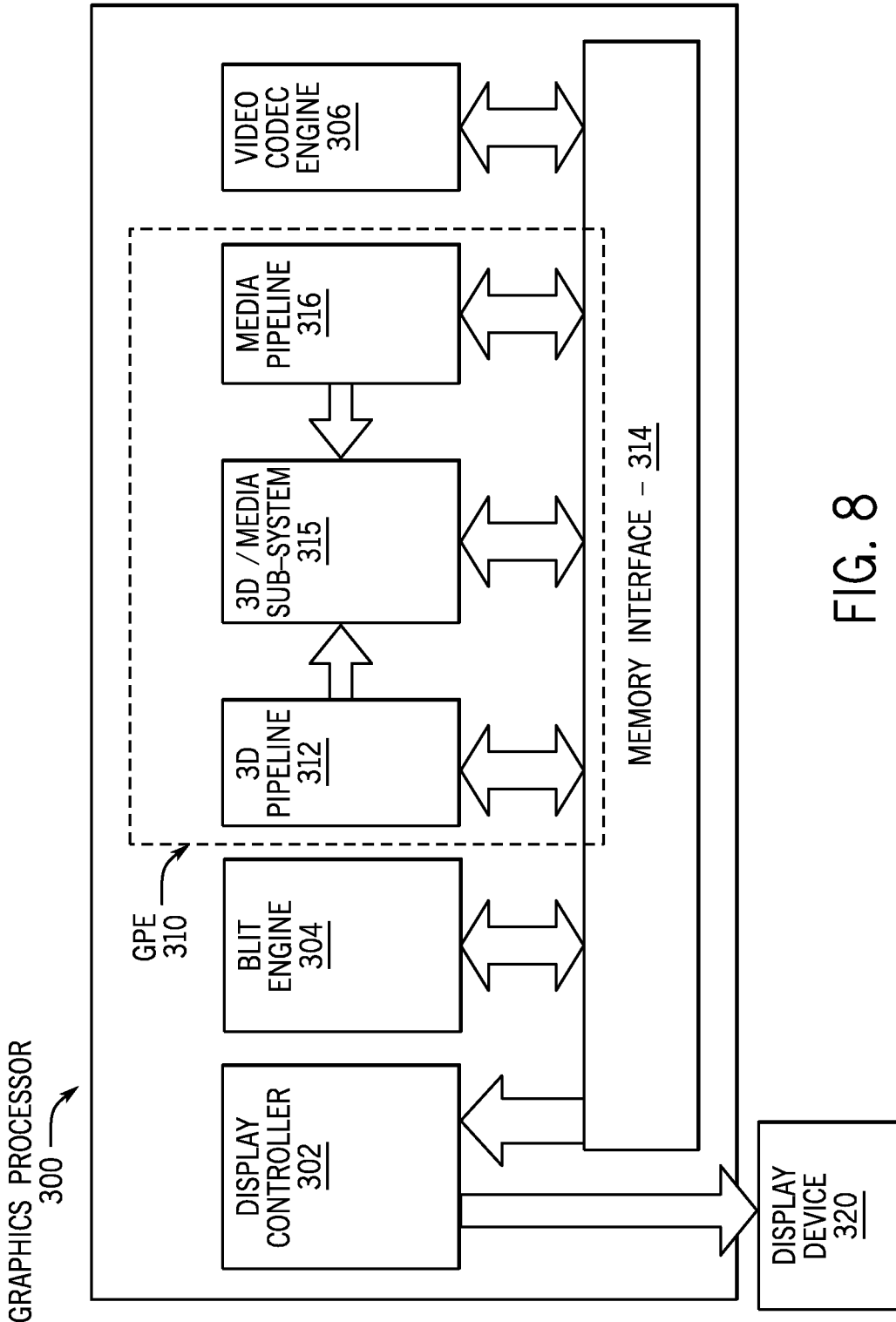


FIG. 8

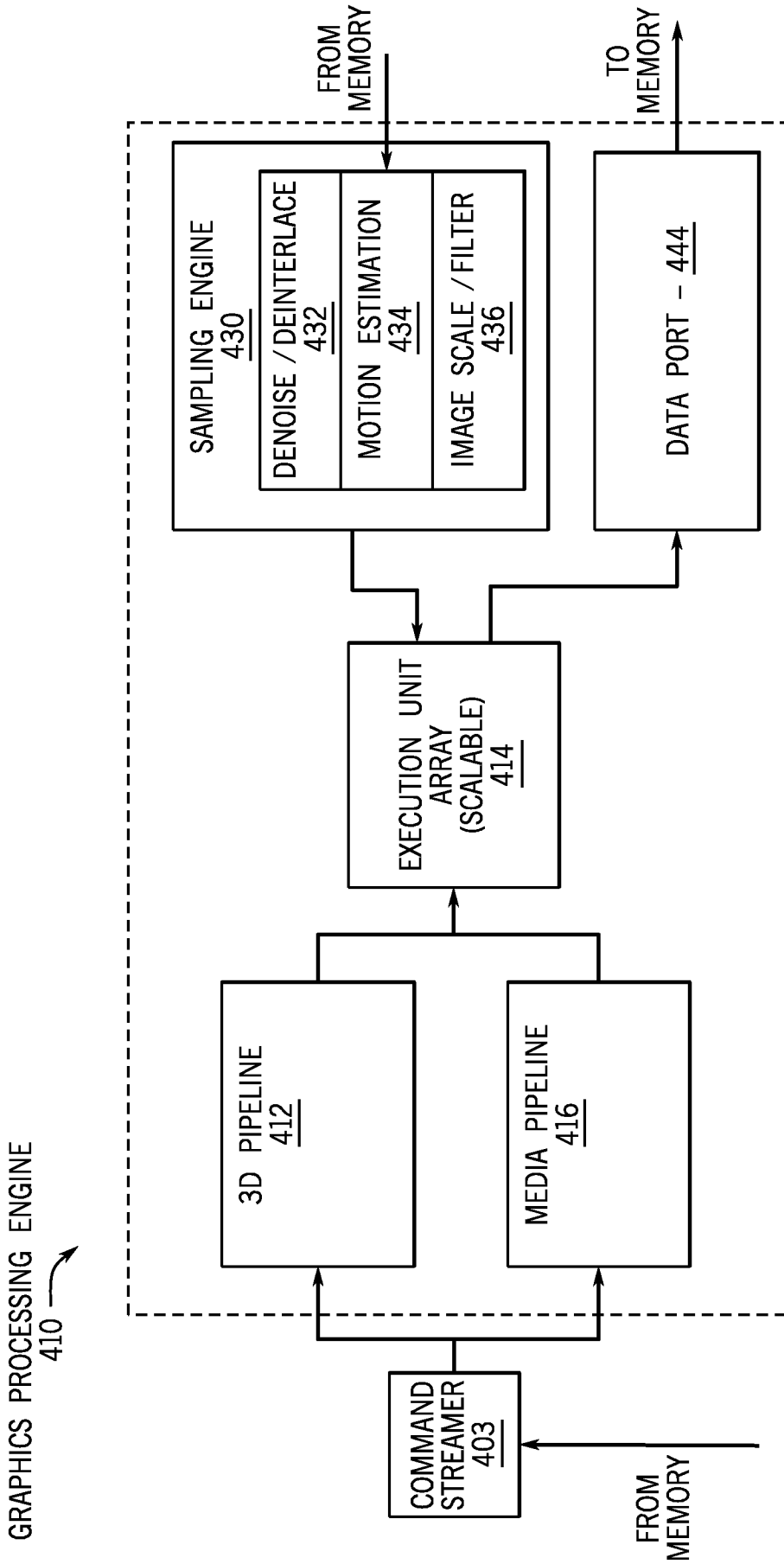


FIG. 9

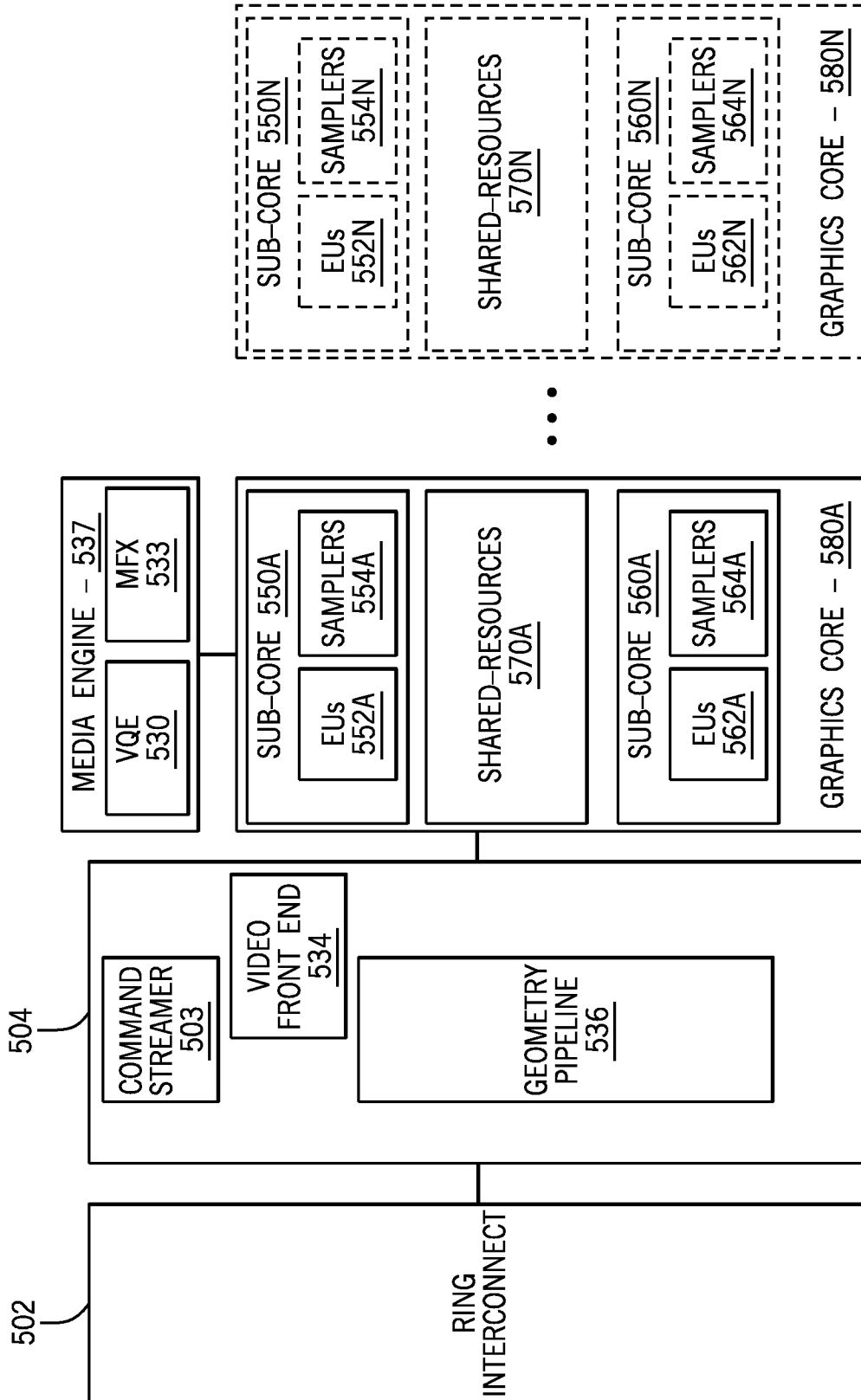


FIG. 10

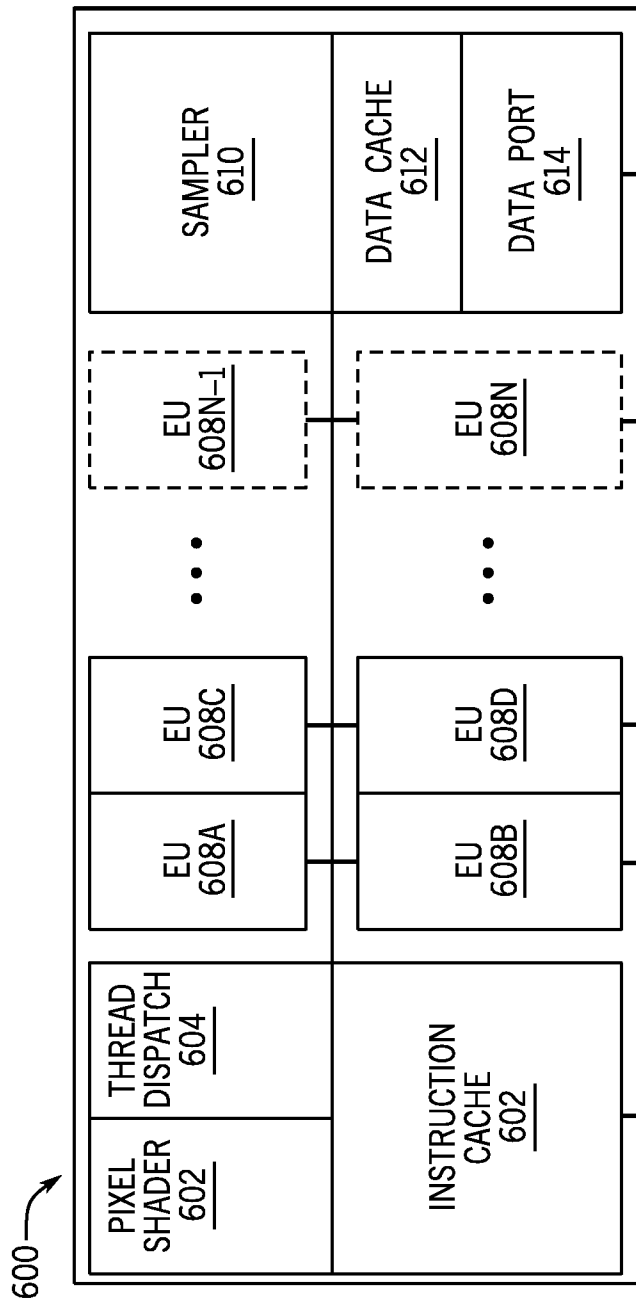


FIG. 11

GRAPHICS CORE INSTRUCTION FORMATS

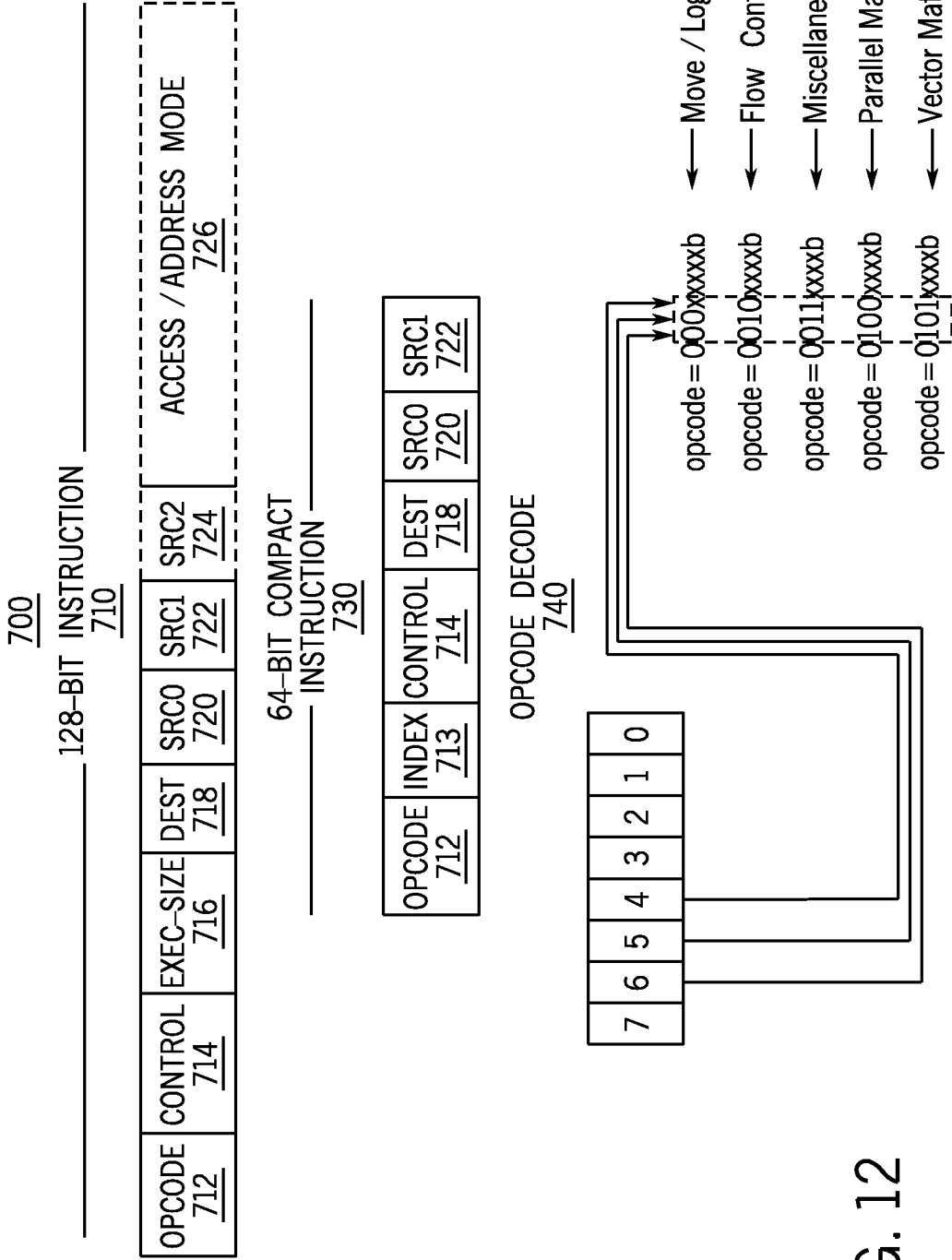


FIG. 12

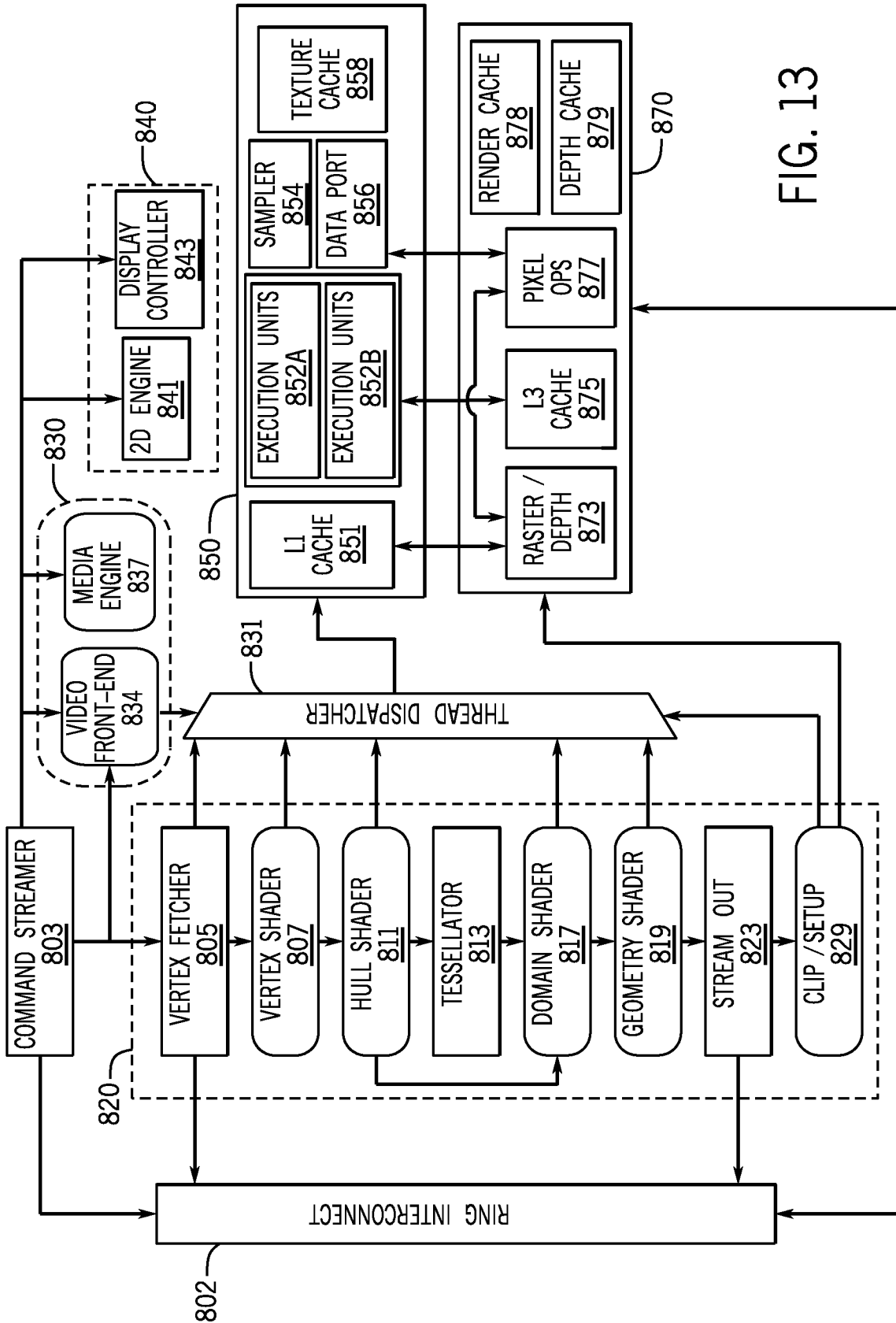


FIG. 13

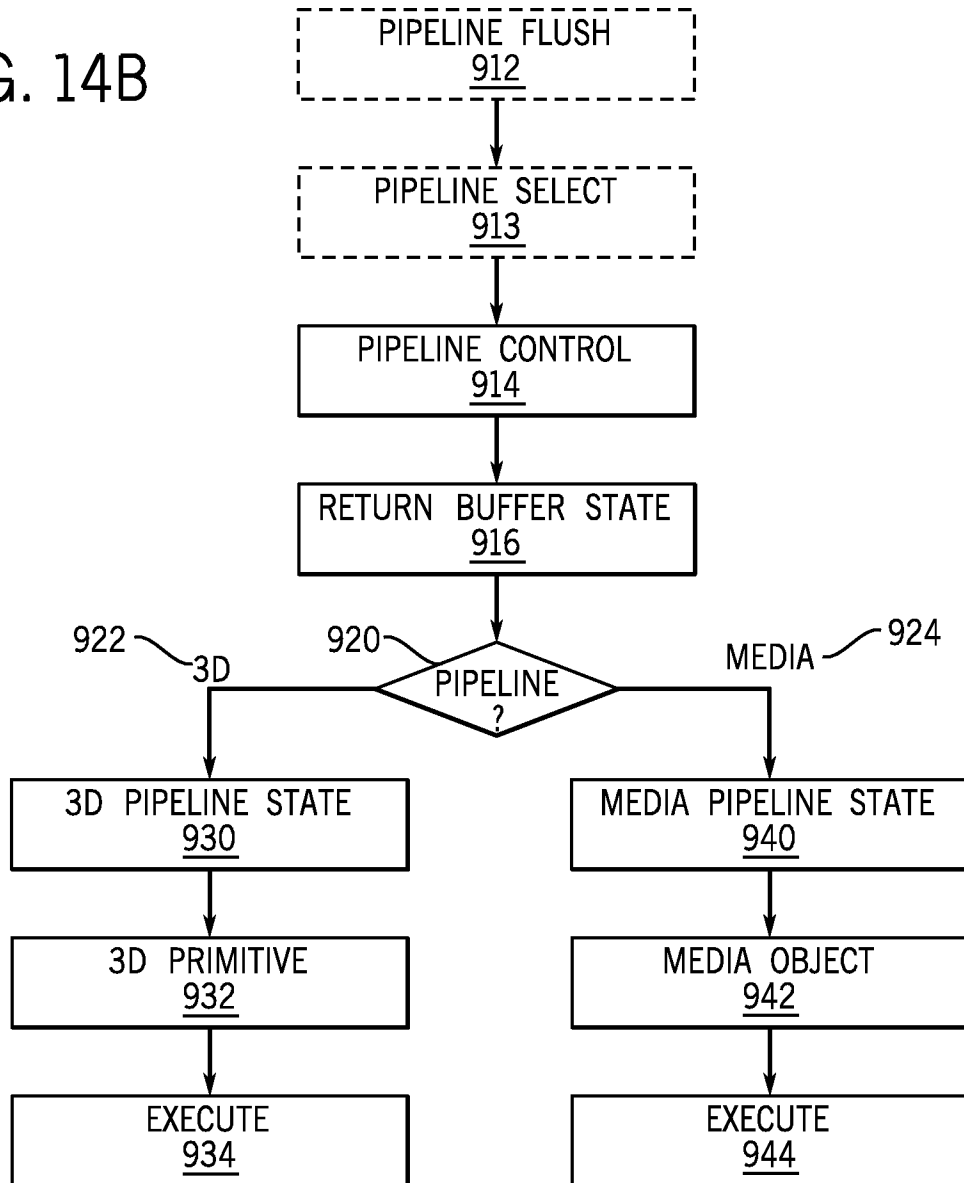
FIG. 14A

SAMPLE COMMAND
FORMAT
900



FIG. 14B

SAMPLE COMMAND SEQUENCE
910



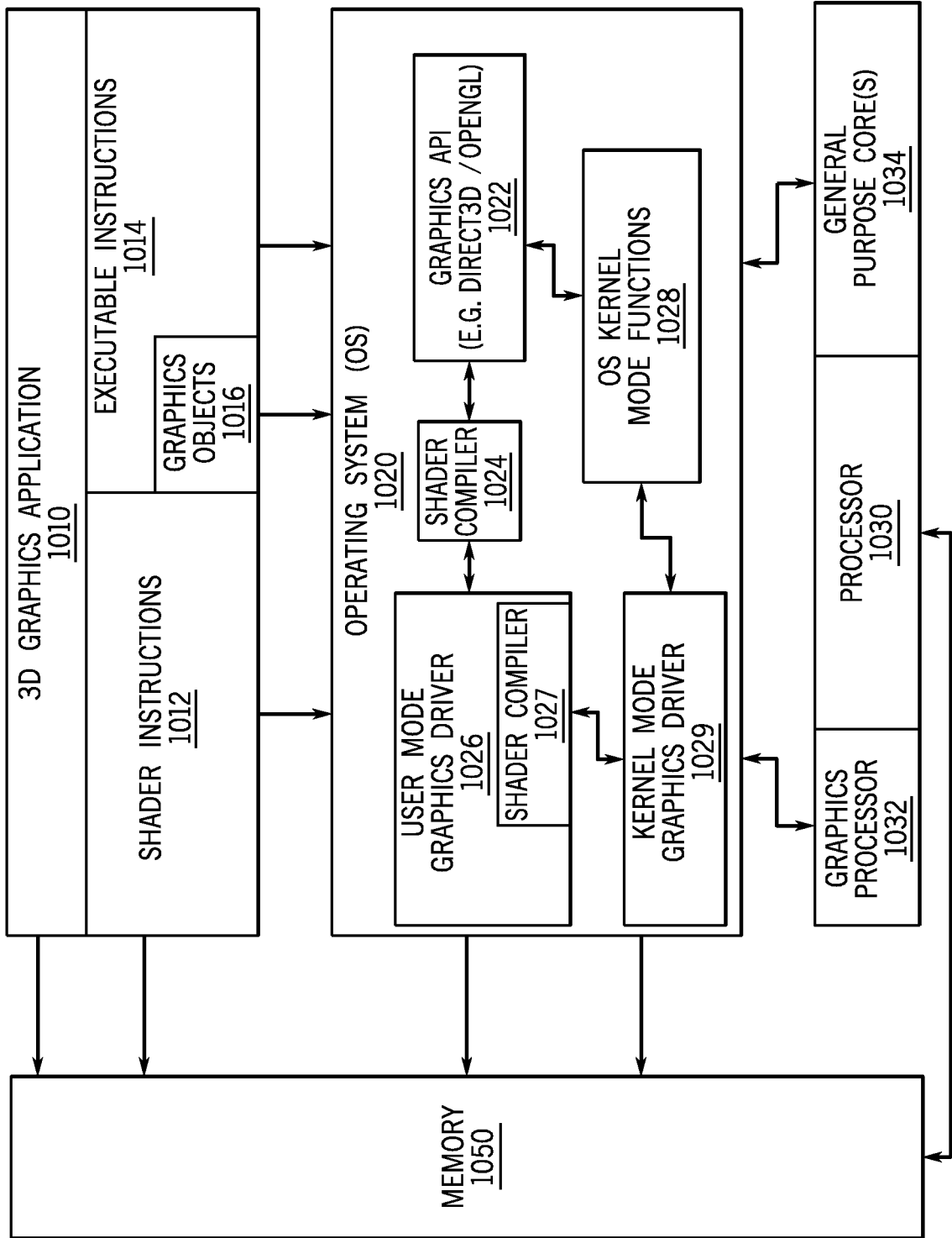


FIG. 15

A. CLASSIFICATION OF SUBJECT MATTER**H04N 19/60(2014.01)i, H04N 19/00(2014.01)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)

H04N 19/60; G09G 5/02; G06F 15/00; H04N 1/60; G06K 9/00; G06K 9/40; G09G 5/10; H04N 19/00

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: convert, RGB, YCbCr, color, tow-dimensional, lookup table.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2014-0002480 A1 (VASUDEV BHASKARAN et al.) 02 January 2014 See paragraphs [0027]-[0030], [0041]-[0043], [0067]; claims 21-26; and figures 1-3.	1-4, 7-9, 11-14, 17-19, 21-24
Y		5-6, 10, 15-16, 20, 25
Y	US 2007-0139448 A1 (ZENGOIANG CHEN et al.) 21 June 2007 See paragraphs [0079]-[0095]; claims 1-2; and figures 1-4.	5-6, 15-16, 25
Y	US 2010-0194773 A1 (HAO PAN et al.) 05 August 2010 See paragraphs [0057], [0133]; claims 1-2; and figure 3.	10, 20
A	US 2011-0149308 A1 (ARIANNE THERESE HINDS et al.) 23 June 2011 See paragraphs [0005]-[0010], [0020]-[0022]; claims 1-2; and figures 1-3.	1-25
A	US 2010-0284614 A1 (XINYU XU et al.) 11 November 2010 See paragraphs [0024]-[0051]; claims 5-8; and figures 1-3.	1-25

 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

"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

30 March 2016 (30.03.2016)

Date of mailing of the international search report

31 March 2016 (31.03.2016)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

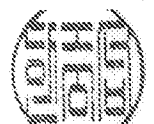
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

HAN, JOONG SUB

Telephone No. +82-42-481-3578



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/056675

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014-0002480 A1	02/01/2014	US 2011-013208 A1 US 8537177 B2 US 8860747 B2 WO 2010-147941 A1	20/01/2011 17/09/2013 14/10/2014 23/12/2010
US 2007-0139448 A1	21/06/2007	CN 1987992 A CN 1987992 C US 7768539 B2 WO 2007-071195 A1	27/06/2007 26/11/2008 03/08/2010 28/06/2007
US 2010-0194773 A1	05/08/2010	US 8238654 B2	07/08/2012
US 2011-0149308 A1	23/06/2011	US 8928936 B2	06/01/2015
US 2010-0284614 A1	11/11/2010	US 8351724 B2 WO 2010-128683 A1	08/01/2013 11/11/2010