



- (51) International Patent Classification:
G06T 15/50 (2006.01) *G06T 15/10* (2006.01)
- (21) International Application Number:
PCT/CN2015/098934
- (22) International Filing Date:
25 December 2015 (25.12.2015)
- (25) Filing Language: English
- (26) Publication Language: English
- (71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95054 (US).
- (72) Inventors; and
(71) Applicants (for US only): CHANG, Yuanzhang [CN/CN]; 9/F Raycom Infotech Park A, Haidian District, Beijing 100190 (CN). JIAO, Shaohui [CN/CN]; 8F, Raycom InfoTech Park A, 2 Kexueyuan South Rd, Beijing 100190 (CN). TONG, Xiaofeng [CN/CN]; Room 1103, Building 1, ShuangyushuXili, Haidian District, Beijing 100086 (CN). LI, Qiang [CN/CN]; 8F, Raycom Infotech Park A No. 2, Kexueyuan South Road, Haidian District, Beijing 100080 (CN). LI, Wenlong [CN/CN]; 4-502 Building 18, Tiantongyuan West 2 District, Changping District, Beijing 102218 (CN).
- (74) Agent: SHANGHAI PATENT & TRADEMARK LAW OFFICE, LLC; 435 Guiping Road, Shanghai 200233 (CN).
- (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,

[Continued on next page]

(54) Title: UNIFIED ENVIRONMENTAL MAPPING FRAMEWORK

(57) Abstract: Methods and apparatus relating to a unified environmental mapping framework are described. In an embodiment, Environmental Mapping (EM) logic performs one or more operations to extract illumination information for an object from an environmental map in response to a determination that the object has a diffuse surface and/or specular surface. Memory, coupled to the EM logic, stores data corresponding to the environmental map. Other embodiments are also disclosed and claimed.

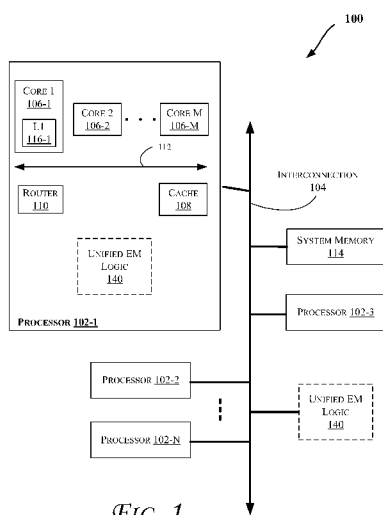


FIG. 1



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

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

DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT,
LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE,
SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA,
GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

UNIFIED ENVIRONMENTAL MAPPING FRAMEWORK

FIELD

[0001] The present disclosure generally relates to the field of electronics. More particularly, an embodiment relates to a unified environmental mapping framework.

BACKGROUND

[0002] In three dimensional (3D) graphics programs, Environmental Mapping (EM) may be used to provide an efficient image-based rendering technique for approximating the appearance of a reflective surface with a precomputed environment map. The environment map stores the image of surrounding environment of the rendered object. However, EM solutions are generally used on specular surface material and are targeted to execute on a relatively powerful computer (e.g., for professional designers) instead of a mobile platform.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The detailed description is provided with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different figures indicates similar or identical items.

[0004] Figs. 1, 6, 7, 16, and 18 illustrate block diagrams of embodiments of computing systems, which may be utilized to implement various embodiments discussed herein.

[0005] Figs. 2A, 2B, 2C, 2D, 2E, 3, and 4 illustrate sample images and/or image modifications as further discussed herein, according to some embodiments.

[0006] Fig. 5 illustrates a flow diagram of a method according to an embodiment.

[0007] Figs. 8-12 and 14 illustrate various components of processors in accordance with some embodiments.

[0008] Fig. 13 illustrates graphics core instruction formats, according to some embodiments.

[0009] Figs. 15A and 15B illustrate graphics processor command format and sequence, respectively, according to some embodiments.

[0010] Fig. 17 illustrates a diagram of IP core development according to an embodiment.

DETAILED DESCRIPTION

[0011] In the following description, numerous specific details are set forth in order to provide a thorough understanding of various embodiments. However, various embodiments may be practiced without the specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as not to obscure the particular embodiments. Further, various aspects of embodiments may be performed using various means, such as integrated semiconductor circuits (“hardware”), computer-readable instructions organized into one or more programs (“software”), or some combination of hardware and software. For the purposes of this disclosure reference to “logic” shall mean either hardware, software, firmware, or some combination thereof.

[0012] As mentioned above, some EM solutions are mostly used for specular surface material and targeted to execute on a powerful computer instead of a mobile platform. For example, some 3D game engines, such as Unity3D™ or OGRE™, are popular for creating high quality rendering results. However, these tools only support specular EM and are suited for professional designers.

[0013] To this end, some embodiments relate to a unified environmental mapping framework. More particularly, the unified environmental mapping framework may support rendering of objects with both specular and diffuse surfaces. Moreover, techniques discussed herein can reach (e.g., super) real-time speed on mobile platforms. In order to support the proposed framework on mobile devices, the rendering may be accelerated with GPU (OpenGL™ ES 3.0 shader).

Moreover, a sampling technique may be used (e.g., in the pre-filter operation of Fig. 5) to further speed up calculation(s).

[0014] As discussed herein, in computer graphics, reflection off of smooth surfaces (such as mirrors or a calm body of water) leads to a type of reflection called a “specular” reflection. By contrast, reflection off of rough surfaces (such as clothing, paper, and the asphalt roadway) leads to a type of reflection known as “diffuse” reflection. Also, as discussed herein the terms “EM” generally refers to an “environmental mapping” or “environment mapping” technique, while an “environmental map” or “environment map” generally refers to a rendered image.

[0015] Also, the scenes, images, or frames discussed herein (e.g., which may be processed by graphics logic (e.g., graphics logic 140 of Fig. 1) in various embodiments) may be captured by an image capture device (such as a digital camera (that may be embedded in another device such as a smart phone, a tablet, a laptop, a stand-alone camera, or other mobile devices such as those discussed herein) or an analog device whose captured images are subsequently converted to digital form). Moreover, the image capture device may be capable of capturing multiple frames in an embodiment. Further, a scene may include one or more frames. One or more frames may be designed/generated on a computer in some embodiments. Also, one or more of the frames of the scene may be presented via a display (such as a liquid crystal display, or another type of a flat panel display device, etc.).

[0016] Further, some embodiments may be applied in computing systems that include one or more processors (e.g., with one or more processor cores), such

as those discussed with reference to Figs. 1-18, including for example mobile computing devices, e.g., a smartphone, tablet, UMPC (Ultra-Mobile Personal Computer), laptop computer, Ultrabook™ computing device, smart devices (such as a smart watch, smart glasses, or a smart bracelet), etc. More particularly, Fig. 1 illustrates a block diagram of a computing system 100, according to an embodiment. The system 100 may include one or more processors 102-1 through 102-N (generally referred to herein as “processors 102” or “processor 102”). The processors 102 may include general-purpose CPUs and/or GPUs in various embodiments. The processors 102 may communicate via an interconnection or bus 104. Each processor may include various components some of which are only discussed with reference to processor 102-1 for clarity. Accordingly, each of the remaining processors 102-2 through 102-N may include the same or similar components discussed with reference to the processor 102-1.

[0017] In an embodiment, the processor 102-1 may include one or more processor cores 106-1 through 106-M (referred to herein as “cores 106,” or “core 106”), a cache 108, and/or a router 110. The processor cores 106 may be implemented on a single Integrated Circuit (IC) chip. Moreover, the chip may include one or more shared and/or private caches (such as cache 108), buses or interconnections (such as a bus or interconnection 112), graphics and/or memory controllers (such as those discussed with reference to Figs. 6-18), or other components.

[0018] In one embodiment, the router 110 may be used to communicate between various components of the processor 102-1 and/or system 100. Moreover,

the processor 102-1 may include more than one router 110. Furthermore, the multitude of routers 110 may be in communication to enable data routing between various components inside or outside of the processor 102-1.

[0019] The cache 108 may store data (e.g., including instructions) that are utilized by one or more components of the processor 102-1, such as the cores 106. For example, the cache 108 may locally cache data stored in a memory 114 for faster access by the components of the processor 102 (e.g., faster access by cores 106). As shown in Fig. 1, the memory 114 may communicate with the processors 102 via the interconnection 104. In an embodiment, the cache 108 (that may be shared) may be a mid-level cache (MLC), a last level cache (LLC), etc. Also, each of the cores 106 may include a level 1 (L1) cache (116-1) (generally referred to herein as “L1 cache 116”) or other levels of cache such as a level 2 (L2) cache. Moreover, various components of the processor 102-1 may communicate with the cache 108 directly, through a bus (e.g., the bus 112), and/or a memory controller or hub.

[0020] As shown in Fig. 1, the processor 102 may further include unified EM logic 140, which is capable of providing a unified EM framework such as discussed herein. Further, logic 140 may have access to one or more storage devices discussed herein (such as video (or image, graphics, etc.) memory, cache 108, L1 cache 116, memory 114, register(s), or another memory in system 100) to store information relating to operations of the logic 140, such as information communicated with various components of system 100 as discussed herein to facilitate provision of the unified EM frame work. Also, while logic 140 is shown

inside the processor 102 (or coupled to interconnection 104), it may be located elsewhere in the system 100 in various embodiments. For example, logic 140 may replace one of the cores 106, may be coupled directly to interconnection 112, etc.

[0021] As mentioned above, some EM solutions are mostly used for specular surface material and targeted to execute on a powerful computer instead of a mobile platform, e.g., suited for professional designers. In addition, the G3D™ engine appears to generally describe a real-time environment lighting method for glossy surfaces. Although the method can be applied on mobile platforms, it is not appropriate for rendering of rough diffuse surface (such as those shown in Figs. 2A(b), 2A(d), 2E(a) and 2E(c)). Moreover, Face rig™ software appears to support EM for both specular and diffuse surface, but it is not usable on mobile devices due to its extremely high computational demands.

[0022] To this end, some embodiments provide a unified environmental mapping framework, which supports rendering of (e.g., 2D or 3D) objects for both specular and diffuse surfaces. Moreover, techniques discussed herein can reach (e.g., super) real-time speed on mobile platforms. In order to support the proposed framework on mobile devices, the rendering can be accelerated with GPU, e.g., using any type of shading or shader support/language such as an OpenGL™ ES 3.0 shader for example. Moreover, a sampling technique can also be used in the pre-filter operation in Fig. 5 to further speed up calculation(s).

[0023] As discussed herein, environment maps are maps which can efficiently assist in finding specular and/or diffuse reflections of surrounding environments given the incident lighting and the normal at a point. Environment

maps are pre-sampled in the form of background images or cube maps in accordance with some embodiments.

[0024] Moreover, with respect to performance, in order to support the proposed unified EM framework on mobile devices, the rendering may be accelerated with a GPU. Specifically, the rendering color of each pixel of the virtual object may be calculated in parallel. For example, OpenGL ES 3.0 shader can be used for the lighting calculation. Moreover, a sampling technique is also used in the pre-filter operation of Fig. 5 to further speed up the calculation(s) as further discussed below. As a result, some embodiments may achieve super real-time speed as shown in Table 1, e.g., 214 fps (or frames per second) on some mobile devices.

Mobile Devices	Galaxy™ S3	Galaxy Note2	iPhone™ 5	iPhone 5s	iPhone 6	iPhone 6plus
Rendering fps	46	52	108	195	214	197

Table 1: Sample average fps values for different mobile devices

[0025] In addition, users may easily customize the object(s) and/or environment(s) to achieve varying effects. Since embodiments provide convenient solutions for user interaction, users are allowed to easily achieve many kinds of rendering results with little effort.

[0026] Rendering examples for the unified EM framework on multiple virtual objects are shown in Fig. 2A. The framework is powerful and useful in many applications (such as animation, movies, virtual reality systems or games, etc.) for displaying vivid cartoon characters with cool/interesting visual experience.

More particularly, Fig. 2A illustrates rendering of virtual objects influenced by an environment, according to some embodiments. For example, Fig. 2A (a, b) illustrate diffuse EM for sample objects as the final body texture is blended with the environment color. Fig. 2A(c) shows specular EM as the robot body is reflective of the surrounding scene. As shown in Fig. 2A(d), the surface material of the fox body is diffuse and the glasses are specular. Fig. 2A(d) shows the unified framework with the fox body texture blended with the green environmental color and the glasses that are reflective of the surrounding forest. Accordingly, with the unified EM framework, users can create a variety of high quality rendering results.

[0027] Furthermore, in contrast to G3D engine, some embodiments can achieve realistic rendering results for both rough diffuse surfaces (see, e.g., Figs. 2A(b), 2A(d), 2E(a) and 2E(c)) and glossy surfaces (see, e.g., Figs 2A(c), 2B(b), and 2B(c)). In addition, in order to achieve real-time performance, some calculation approximations are applied in the G3D engine. Thus, artifacts from the approximations are most visible on completely smooth geometry. In contrast, the results in some embodiments do not include such artifacts, even though some approximation may still be used. Furthermore, G3D engine appears to use a plurality of environment MIP maps for lighting while only one environment map is used in some embodiments. This makes some embodiments more easily user customizable.

[0028] Namely, one embodiment presents an easy-to-use way to let users customize their own rendering results with a unified environmental mapping framework. The rendering is accelerated with a GPU shader. User customization of

object or environment and/or facial gesture recognition may also be included to enhance user interaction and enrich user experience.

[0029] Also, as discussed herein, in 2D/3D computer graphics, mipmaps (also MIP maps) are pre-calculated, optimized sequences of textures, each of which is a progressively lower resolution representation of the same image. Generally, the height and width of each image, or level, in the mipmap is a power of two smaller than the previous level.

[0030] Fig. 2B illustrates sample rendering image results, according to some embodiments. More particularly, Fig. 2B(a) shows a classic Phong shading model image result, Fig. 2B(b) shows a diffuse EM image result, and Fig. 2B(c) shows a specular EM image result.

[0031] Moreover, classic shading algorithms are generally not realistic enough for high quality rendering applications. The popular Phong shading or interpolation model is a representative model. Classic shading methods consider only the direct light energy transfer between a light source and an object surface. The rendering results are not realistic enough for applications with high quality rendering requirement. Physically based global illumination algorithms calculate light energy interactions, including both direct and indirect lighting, between all light intersected surfaces in an environment. Although global illumination algorithms can provide photo-realistic rendering results, the computational cost of these methods are too large to be used on mobile devices.

[0032] In accordance with some embodiments, the unified EM framework provides much faster results than physically based global illumination algorithms

with similar rendering effects. Although it may have nearly the same speed as the classic Phong shading model, the rendering results are much better as shown in Fig. 2B. The classic Phong shading model is used in Fig. 2B(a). In Fig. 2B(b), the top part of the model's head is influenced by the sky color and the bottom part is very close to the sand color. In Fig. 2B(c), the surface material is specular and reflective of the blue sky. It can be easily observed that the rendering results of Fig. 2B(b) and 2B(c) are more realistic and immersive with the surrounding environment when techniques in accordance with some embodiments are applied.

[0033] As mentioned above, the unified EM framework may achieve super real-time performance with GPU acceleration. It can be easily applied on various platforms, including both PC and mobile devices. In mobile implementation, OpenGL ES 3.0 shading language may be used for acceleration. Moreover, a sampling method is also used in the pre-filter operation of Fig. 5 to further speed up the calculation(s). Table 1 shows the average FPS on different mobile devices.

[0034] Various embodiments can be applied in different situations such as in the following cases: (1) stationary environments (see, e.g., Fig. 2C); (2) varying environments (see, e.g., Fig. 2D); (3) user customization (see, e.g., Fig. 2E); (4) vision guided rendering (see, e.g., Fig. 3); and/or (5) scalability on various platforms (see, e.g., Fig. 4).

[0035] More particularly, Fig. 2C illustrates rendering comparisons for different environments, when diffuse EM is adopted in accordance with an embodiment. As shown in Fig. 2C, the rendering effect changes with the

environment. Diffuse EM is taken as an example and specular EM can be done in a very similar manner.

[0036] Fig. 2D illustrates the environment changes in clockwise rotation, where influence on the object can be easily observed, according to some embodiments. The environment can also be changed seamlessly. For Fig. 2D, a cube map can be used in which six environments are projected onto the six faces of a cube. The cube map is in clockwise rotation. It can be easily observed that the object surface rendering varies smoothly when the environment changes.

[0037] Fig. 2E illustrates that a user can customize the object and environment, and the object rendering will be influenced by the new environment, according to some embodiments. For example, users can interact with I/O (Input/Output) devices (such as touch pad, mouse, pen, etc.) of a PC or touch screen (or pen) of mobile device to customize the virtual object or environment as shown in Fig. 2E. Then, the object rendering will be updated accordingly. The object can be a 3D model created with professional software or scanned from the real world. The environment map can also be chosen from downloaded images, user-drawn pictures or photos. With such user customizations, the overall user experience can be enriched (and/or user interaction can be enhanced as well).

[0038] Fig. 3 illustrates sample facial gestures that can be detected with a facial gesture recognition algorithm and utilized to drive the object movements with object rendering updated, according to some embodiments. The proposed system allows direct manipulation to specify and control the rendering results with facial gestures tracked with facial trackers and captured with a camera. Hence, in

an embodiment, facial gestures are recognized with a facial tracking algorithm to control the object movements as shown in Fig. 3. The facial gestures can include opening and closing of eyes or mouth, head movements and so on.

[0039] Fig. 4 illustrates application of an embodiment on mobile platforms, e.g., a smart device with super real-time performance. Such techniques can be applied on stationary computing devices as well as on mobile computing devices (e.g., with real-time performance) as shown in Fig. 4. Users can utilize these techniques to create many interesting and/or creative animations for daily use.

[0040] Fig. 5 illustrates a flow diagram of a method 500 to provide a unified EM framework, according to some embodiments. In one embodiment, method 500 shows operations performed by logic 140 to provide the unified environmental mapping framework such as discussed herein. In an embodiment, various components discussed with reference to the other figures may be utilized to perform one or more of the operations of method 500.

[0041] Referring to Figs. 1-5, at operation 502, method 500 starts by loading an (e.g., entire) animation scene, or more generally one or more frames of the scene to be processed. By way of example, the loaded scene may include 3D virtual objects, model textures, light distribution and surrounding environment, and/or environment maps. Users may be allowed to customize 3D virtual objects with 3D scanning from real world or models created with professional software. The environment map may also be chosen from downloaded images, user-drawn pictures, and/or photos. In one embodiment, a background image (e.g., in JPEG

(Joint Photographic Experts Group) format or another format) is used for the specular or diffuse environment map.

[0042] Operation 504 determines the material for an object under processing (e.g., from the loaded scene of operation 502). Moreover, in order to provide realistic rendering results for the virtual objects, object lighting may be calculated by taking into consideration object surface material and/or scene light distribution parameter(s). Surface material properties can detail one or more of a material's diffuse reflection, ambient reflection, and/or specular reflection characteristics. As discussed herein, diffuse reflection is the reflection of light from a surface such that an incident ray is reflected at many angles and the surface will have equal luminance in all directions.

[0043] Further, specular reflection is a mirror-like reflection from a surface, in which light from a single incident direction is reflected into a narrow band of reflected directions. In an embodiment, the ambient reflection is only dependent on the object surface material and is uniform in all directions. Calculation of the diffuse and specular reflection may be focused since ambient reflection is a constant value in an embodiments. Examples are shown in the rightmost figures of Figs. 2E and 4 as the material for the fox body is diffuse while the glasses are specular.

[0044] If operation 504 determines the object material to be diffuse material, operation 506 may perform pre-filter operation(s) further discussed below. More particularly, pre-filter operation(s) may be performed at 506 to extract the illumination information from a diffuse environmental map. In one embodiment, a

720x1280 resolution rectangular background image in JPEG format is used to represent the original square environment map. Using an analytic expression for the irradiance in terms of spherical harmonic coefficients of the lighting, nine coefficients L_{lm} for $l \leq 2$, corresponding to the lowest-frequency modes of the illumination, are computed. Each color channel is treated separately, so the coefficients can be thought of as RGB (Red, Green, Blue) values:

$$L_{lm} = \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} L(\theta, \phi) Y_{lm}(\theta, \phi) \sin\theta d\theta d\phi \quad (1)$$

[0045] The expressions for Y_{lm} in equation (1) may be pre-calculated values. The integrals can provide sums of the pixels in the environment map, weighted by the function Y_{lm} . The integrals can also be viewed as moments of the lighting, or as inner-products of the function L and Y_{lm} .

[0046] Additionally, it may be inefficient to calculate equation (1) for each pixel of the environment map on a mobile device. Thus, a sampling operation is added to accelerate the performance in an embodiment. Adaptive sampling based on pixel importance is one candidate. More particularly, pixels on image contours or boundaries are regarded as pixels with high importance and may be sampled with higher priority. In some embodiments, the rendering results of uniform sampling with constant pixel interval in both UV (where letters “U” and “V” denote the axes of a two-dimensional texture) directions may be considered to be quite realistic and/or provide relatively high computational efficiency.

[0047] After operation 506 (or determination at operation 504 that the object material is specular), operation 508 performs scene/object rendering on a GPU (such as any of the GPUs discussed herein). More particularly, for each frame,

vertices of the objects are stored as one vertex buffer object for GPU processing. OpenGL ES 3.0 shader is used for GPU calculation(s) at operation 508 in one embodiment. For specular EM, the reflection is implemented firstly by calculating the vector that the object is being viewed at. This camera ray c is reflected with the surface normal n , where the camera vector intersects the object. This results in a reflected ray $\text{reflect}(c, n)$ which is then passed to the specular environment map to determine the reflection color C_r used in the rendering calculation. This creates an effect that the object is reflective. The color C_r is given by:

$$C_r = \text{texCube}(\text{environmentMap}, \text{reflect}(c, n)) \quad (2)$$

[0048] The final rendering color is a linear interpolation of the color C_r and diffuse map color $\text{texture2D}(\text{diffMap}, \text{textCoord})$ with an interpolation parameter refPar :

$$\text{glFragColor} = \text{mix}(\text{color} * \text{texture2D}(\text{diffMap}, \text{textCoord}), C_r, \text{refPar}) \quad (3)$$

[0049] In equations (3), the variable color may be a color calculated by using the Phong shading model. For diffuse EM, the rendering equation is

$$B_i(p, n) = \rho(p) n^t M_i n \quad (4)$$

[0050] In equation (4), the term $B_i(p, n)$ corresponds directly to the image color intensity with subscript i showing the RGB index. $\rho(p)$ is the surface albedo (also referred to as surface reflection coefficient) dependent on position p , where $n^t = (x, y, z, 1)$ is the normalized surface normal. M_i is a symmetric 4×4 matrix. Each color RGB has an independent matrix M_i . The radiosity $B_i(p, n)$ is a quadratic polynomial of the coordinate of the surface normal. The entries of matrix M_i depend on the first nine moments of L_{lm} .

[0051] Moreover, another innovative extension may be made in an embodiments. More specifically, as shown in Fig. 2D, a cube map is used for diffuse EM and it is in clockwise or anti-clockwise rotation, the object rendering can be updated per frame. Thus, the final matrix M_i may be updated with linear interpolation of the matrices of adjacent faces:

$$M_i = \alpha M_{ai} + (1 - \alpha) M_{bi} \quad (5)$$

[0052] In equation (5), α ($0 \leq \alpha \leq 1$) is a coefficient with linear relationship to the rotation angle, M_{ai} and M_{bi} are the calculated matrices of adjacent face a and face b correspondingly. Finally, the calculated RGB values $B_i(p, n) \ i \in \{RGB\}$ for diffuse EM are added as an ambient part of the Phong shading model to provide a total color value, which may be further combined with surface textures to determine the final surface color $gl_{FragColor}$.

[0053] At operation 510, the scene and/or object under processing are updated. For example, as shown in Fig. 5, users can interact with I/O devices of a computing device (such as a touch screen, mouse, keyboard, etc.) to customize the object or environment as shown in Fig. 2E. Then, the rendering result of object will be updated accordingly at operation 510. In addition, facial gestures may also be detected with a facial gesture recognition algorithm to drive the object movements as shown in Fig. 3. At operation 512, method 500 determines whether any more objects or scenes are remaining to be processed and returns to operation 502 if applicable.

[0054] Accordingly, some embodiments provide a novel real-time unified environmental mapping framework which can realistically render virtual objects

with both specular and diffuse surfaces to enrich the user experience of interacting with virtual object in 3D scenes. Also, various embodiments may be applied in animation or rendering tools/software, mobile/non-mobile computing device game engines, such as plug-in in VR/AR (Virtual Reality/) systems or other 3D applications.

[0055] In some embodiments, one or more of the components discussed herein can be embodied as a System On Chip (SOC) device. Fig. 6 illustrates a block diagram of an SOC package in accordance with an embodiment. As illustrated in Fig. 6, SOC 602 includes one or more Central Processing Unit (CPU) cores 620 (which may be the same as or similar to the cores 106 of Fig. 1), one or more Graphics Processor Unit (GPU) cores 630 (which may be the same as or similar to the logic 140 of Fig. 1), an Input/Output (I/O) interface 640, and a memory controller 642. Various components of the SOC package 602 may be coupled to an interconnect or bus such as discussed herein with reference to the other figures. Also, the SOC package 602 may include more or less components, such as those discussed herein with reference to the other figures. Further, each component of the SOC package 620 may include one or more other components, e.g., as discussed with reference to the other figures herein. In one embodiment, SOC package 602 (and its components) is provided on one or more Integrated Circuit (IC) die, e.g., which are packaged into a single semiconductor device.

[0056] As illustrated in Fig. 6, SOC package 602 is coupled to a memory 660 (which may be similar to or the same as memory discussed herein with reference to the other figures such as system memory 114 of Fig. 1) via the

memory controller 642. In an embodiment, the memory 660 (or a portion of it) can be integrated on the SOC package 602.

[0057] The I/O interface 640 may be coupled to one or more I/O devices 670, e.g., via an interconnect and/or bus such as discussed herein with reference to other figures. I/O device(s) 670 may include one or more of a keyboard, a mouse, a touchpad, a display, an image/video capture device (such as a camera or camcorder/video recorder), a touch screen, a speaker, or the like. Furthermore, SOC package 602 may include/integrate logic 140 in an embodiment. Alternatively, the logic 140 may be provided outside of the SOC package 602 (i.e., as a discrete logic).

[0058] Fig. 7 is a block diagram of a processing system 700, according to an embodiment. In various embodiments the system 700 includes one or more processors 702 and one or more graphics processors 708 (such as the logic 140 of Fig. 1), and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 702 (such as processor 102 of Fig. 1) or processor cores 707 (such as cores 106 of Fig. 1). In one embodiment, the system 700 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

[0059] An embodiment of system 700 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 some embodiments system 700 is a mobile phone, smart phone, tablet

computing device or mobile Internet device. Data processing system 700 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 some embodiments, data processing system 700 is a television or set top box device having one or more processors 702 and a graphical interface generated by one or more graphics processors 708.

[0060] In some embodiments, the one or more processors 702 each include one or more processor cores 707 to process instructions which, when executed, perform operations for system and user software. In some embodiments, each of the one or more processor cores 707 is configured to process a specific instruction set 709. In some embodiments, instruction set 709 may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). Multiple processor cores 707 may each process a different instruction set 709, which may include instructions to facilitate the emulation of other instruction sets. Processor core 707 may also include other processing devices, such as a Digital Signal Processor (DSP).

[0061] In some embodiments, the processor 702 includes cache memory 704. Depending on the architecture, the processor 702 can have a single internal cache or multiple levels of internal cache. In some embodiments, the cache memory is shared among various components of the processor 702. In some embodiments, the processor 702 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among processor cores 707 using known cache coherency techniques. A register file 706 is additionally included in

processor 702 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 702.

[0062] In some embodiments, processor 702 is coupled to a processor bus 710 to transmit communication signals such as address, data, or control signals between processor 702 and other components in system 700. In one embodiment the system 700 uses an exemplary ‘hub’ system architecture, including a memory controller hub 716 and an Input Output (I/O) controller hub 730. A memory controller hub 716 facilitates communication between a memory device and other components of system 700, while an I/O Controller Hub (ICH) 730 provides connections to I/O devices via a local I/O bus. In one embodiment, the logic of the memory controller hub 716 is integrated within the processor.

[0063] Memory device 720 can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as process memory. In one embodiment the memory device 720 can operate as system memory for the system 700, to store data 722 and instructions 721 for use when the one or more processors 702 executes an application or process. Memory controller hub 716 also couples with an optional external graphics processor 712, which may communicate with the one or more graphics processors 708 in processors 702 to perform graphics and media operations.

[0064] In some embodiments, ICH 730 enables peripherals to connect to memory device 720 and processor 702 via a high-speed I/O bus. The I/O peripherals include, but are not limited to, an audio controller 746, a firmware interface 728, a wireless transceiver 726 (e.g., Wi-Fi, Bluetooth), a data storage device 724 (e.g., hard disk drive, flash memory, etc.), and a legacy I/O controller 740 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to the system. One or more Universal Serial Bus (USB) controllers 742 connect input devices, such as keyboard and mouse 744 combinations. A network controller 734 may also couple to ICH 730. In some embodiments, a high-performance network controller (not shown) couples to processor bus 710. It will be appreciated that the system 700 shown is exemplary and not limiting, as other types of data processing systems that are differently configured may also be used. For example, the I/O controller hub 730 may be integrated within the one or more processor 702, or the memory controller hub 716 and I/O controller hub 730 may be integrated into a discreet external graphics processor, such as the external graphics processor 712.

[0065] Fig. 8 is a block diagram of an embodiment of a processor 800 having one or more processor cores 802A-802N, an integrated memory controller 814, and an integrated graphics processor 808. The processor 800 may be similar to or the same as the processor 102 discussed with reference to Fig. 1. Those elements of Fig. 8 having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such. Processor 800 can include additional cores up to and including additional core 802N represented by the dashed lined boxes. Each of processor cores 802A-802N includes one or more internal cache

units 804A-804N. In some embodiments each processor core also has access to one or more shared cached units 806.

[0066] The internal cache units 804A-804N and shared cache units 806 represent a cache memory hierarchy within the processor 800. The cache memory hierarchy may include at least one level of instruction and data cache within each processor 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 LLC. In some embodiments, cache coherency logic maintains coherency between the various cache units 806 and 804A-804N.

[0067] In some embodiments, processor 800 may also include a set of one or more bus controller units 816 and a system agent core 810. The one or more bus controller units 816 manage a set of peripheral buses, such as one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express). System agent core 810 provides management functionality for the various processor components. In some embodiments, system agent core 810 includes one or more integrated memory controllers 814 to manage access to various external memory devices (not shown).

[0068] In some embodiments, one or more of the processor cores 802A-802N include support for simultaneous multi-threading. In such embodiment, the system agent core 810 includes components for coordinating and operating cores 802A-802N during multi-threaded processing. System agent core 810 may additionally include a power control unit (PCU), which includes logic and

components to regulate the power state of processor cores 802A-802N and graphics processor 808.

[0069] In some embodiments, processor 800 additionally includes graphics processor 808 to execute graphics processing operations. In some embodiments, the graphics processor 808 couples with the set of shared cache units 806, and the system agent core 810, including the one or more integrated memory controllers 814. In some embodiments, a display controller 811 is coupled with the graphics processor 808 to drive graphics processor output to one or more coupled displays. In some embodiments, display controller 811 may be a separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 808 or system agent core 810.

[0070] In some embodiments, a ring based interconnect unit 812 is used to couple the internal components of the processor 800. 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 some embodiments, graphics processor 808 couples with the ring interconnect 812 via an I/O link 813.

[0071] The exemplary I/O link 813 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 818, such as an eDRAM (or embedded DRAM) module. In some embodiments, each of the processor cores 802-802N and

graphics processor 808 use embedded memory modules 818 as a shared Last Level Cache.

[0072] In some embodiments, processor cores 802A-802N are homogenous cores executing the same instruction set architecture. In another embodiment, processor cores 802A-802N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 802A-802N 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. In one embodiment processor cores 802A-802N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. Additionally, processor 800 can be implemented on one or more chips or as an SoC integrated circuit having the illustrated components, in addition to other components.

[0073] Fig. 9 is a block diagram of a graphics processor 900, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. The graphics processor 900 may be similar to or the same as the logic 140 discussed with reference to Fig. 1. In some embodiments, the graphics processor communicates via a memory mapped I/O interface to registers on the graphics processor and with commands placed into the processor memory. In some embodiments, graphics processor 900 includes a memory interface 914 to access memory. Memory interface 914 can be an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0074] In some embodiments, graphics processor 900 also includes a display controller 902 to drive display output data to a display device 920. Display controller 902 includes hardware for one or more overlay planes for the display and composition of multiple layers of video or user interface elements. In some embodiments, graphics processor 900 includes a video codec engine 906 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.

[0075] In some embodiments, graphics processor 900 includes a block image transfer (BLIT) engine 904 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in one embodiment, 8D graphics operations are performed using one or more components of graphics processing engine (GPE) 910. In some embodiments, graphics processing engine 910 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0076] In some embodiments, GPE 910 includes a 3D pipeline 912 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 912 includes programmable and fixed function elements that perform various tasks within the element and/or spawn execution

threads to a 3D/Media sub-system 915. While 3D pipeline 912 can be used to perform media operations, an embodiment of GPE 910 also includes a media pipeline 916 that is specifically used to perform media operations, such as video post-processing and image enhancement.

[0077] In some embodiments, media pipeline 916 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 video codec engine 906. In some embodiments, media pipeline 916 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 915. The spawned threads perform computations for the media operations on one or more graphics execution units included in 3D/Media sub-system 915.

[0078] In some embodiments, 3D/Media subsystem 915 includes logic for executing threads spawned by 3D pipeline 912 and media pipeline 916. In one embodiment, the pipelines send thread execution requests to 3D/Media subsystem 915, 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 some embodiments, 3D/Media subsystem 915 includes one or more internal caches for thread instructions and data. In some embodiments, the subsystem also includes shared memory, including registers and addressable memory, to share data between threads and to store output data.

[0079] Fig. 10 is a block diagram of a graphics processing engine 1010 of a graphics processor in accordance with some embodiments. In one embodiment, the GPE 1010 is a version of the GPE 910 shown in Fig. 9. Elements of Fig. 10 having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

[0080] In some embodiments, GPE 1010 couples with a command streamer 1003, which provides a command stream to the GPE 3D and media pipelines 1012, 1016. In some embodiments, command streamer 1003 is coupled to memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In some embodiments, command streamer 1003 receives commands from the memory and sends the commands to 3D pipeline 1012 and/or media pipeline 1016. The commands are directives fetched from a ring buffer, which stores commands for the 3D and media pipelines 1012, 1016. In one embodiment, the ring buffer can additionally include batch command buffers storing batches of multiple commands. The 3D and media pipelines 1012, 1016 process the commands by performing operations via logic within the respective pipelines or by dispatching one or more execution threads to an execution unit array 1014. In some embodiments, execution unit array 1014 is scalable, such that the array includes a variable number of execution units based on the target power and performance level of GPE 1010.

[0081] In some embodiments, a sampling engine 1030 couples with memory (e.g., cache memory or system memory) and execution unit array 1014. In some

embodiments, sampling engine 1030 provides a memory access mechanism for execution unit array 1014 that allows execution array 1014 to read graphics and media data from memory. In some embodiments, sampling engine 1030 includes logic to perform specialized image sampling operations for media.

[0082] In some embodiments, the specialized media sampling logic in sampling engine 1030 includes a de-noise/de-interlace module 1032, a motion estimation module 1034, and an image scaling and filtering module 1036. In some embodiments, de-noise/de-interlace module 1032 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 removes data noise from video and image data. In some embodiments, 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 some embodiments, the de-noise/de-interlace module 1032 includes dedicated motion detection logic (e.g., within the motion estimation engine 1034).

[0083] In some embodiments, motion estimation engine 1034 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 some embodiments, a graphics processor media codec uses video motion estimation engine 1034 to perform operations on video at the macro-block level that may

otherwise be too computationally intensive to perform with a general-purpose processor. In some embodiments, motion estimation engine 1034 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.

[0084] In some embodiments, image scaling and filtering module 1036 performs image-processing operations to enhance the visual quality of generated images and video. In some embodiments, scaling and filtering module 1036 processes image and video data during the sampling operation before providing the data to execution unit array 1014.

[0085] In some embodiments, the GPE 1010 includes a data port 1044, which provides an additional mechanism for graphics subsystems to access memory. In some embodiments, data port 1044 facilitates memory access for operations including render target writes, constant buffer reads, scratch memory space reads/writes, and media surface accesses. In some embodiments, data port 1044 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 some embodiments, threads executing on an execution unit in execution unit array 1014 communicate with the data port by exchanging messages via a data distribution interconnect that couples each of the sub-systems of GPE 1010.

[0086] Fig. 11 is a block diagram of another embodiment of a graphics processor 1100. Elements of Fig. 11 having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

[0087] In some embodiments, graphics processor 1100 includes a ring interconnect 1102, a pipeline front-end 1104, a media engine 1137, and graphics cores 1180A-1180N. In some embodiments, ring interconnect 1102 couples the graphics processor to other processing units, including other graphics processors or one or more general-purpose processor cores. In some embodiments, the graphics processor is one of many processors integrated within a multi-core processing system.

[0088] In some embodiments, graphics processor 1100 receives batches of commands via ring interconnect 1102. The incoming commands are interpreted by a command streamer 1103 in the pipeline front-end 1104. In some embodiments, graphics processor 1100 includes scalable execution logic to perform 3D geometry processing and media processing via the graphics core(s) 1180A-1180N. For 3D geometry processing commands, command streamer 1103 supplies commands to geometry pipeline 1136. For at least some media processing commands, command streamer 1103 supplies the commands to a video front end 1134, which couples with a media engine 1137. In some embodiments, media engine 1137 includes a Video Quality Engine (VQE) 1130 for video and image post-processing and a multi-format encode/decode (MFX) 1133 engine to provide hardware-accelerated media data encode and decode. In some embodiments, geometry pipeline 1136 and

media engine 1137 each generate execution threads for the thread execution resources provided by at least one graphics core 1180A.

[0089] In some embodiments, graphics processor 1100 includes scalable thread execution resources featuring modular cores 1180A-1180N (sometimes referred to as core slices), each having multiple sub-cores 1150A-1150N, 1160A-1160N (sometimes referred to as core sub-slices). In some embodiments, graphics processor 1100 can have any number of graphics cores 1180A through 1180N. In some embodiments, graphics processor 1100 includes a graphics core 1180A having at least a first sub-core 1150A and a second core sub-core 1160A. In other embodiments, the graphics processor is a low power processor with a single sub-core (e.g., 1150A). In some embodiments, graphics processor 1100 includes multiple graphics cores 1180A-1180N, each including a set of first sub-cores 1150A-1150N and a set of second sub-cores 1160A-1160N. Each sub-core in the set of first sub-cores 1150A-1150N includes at least a first set of execution units 1152A-1152N and media/texture samplers 1154A-1154N. Each sub-core in the set of second sub-cores 1160A-1160N includes at least a second set of execution units 1162A-1162N and samplers 1164A-1164N. In some embodiments, each sub-core 1150A-1150N, 1160A-1160N shares a set of shared resources 1170A-1170N. In some embodiments, 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.

[0090] Fig. 12 illustrates thread execution logic 1200 including an array of processing elements employed in some embodiments of a GPE. Elements of Fig.

12 having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

[0091] In some embodiments, thread execution logic 1200 includes a pixel shader 1202, a thread dispatcher 1204, instruction cache 1206, a scalable execution unit array including a plurality of execution units 1208A-1208N, a sampler 1210, a data cache 1212, and a data port 1214. In one embodiment the included components are interconnected via an interconnect fabric that links to each of the components. In some embodiments, thread execution logic 1200 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 1206, data port 1214, sampler 1210, and execution unit array 1208A-1208N. In some embodiments, each execution unit (e.g. 1208A) is an individual vector processor capable of executing multiple simultaneous threads and processing multiple data elements in parallel for each thread. In some embodiments, execution unit array 1208A-1208N includes any number individual execution units.

[0092] In some embodiments, execution unit array 1208A-1208N is primarily used to execute “shader” programs. In some embodiments, the execution units in array 1208A-1208N 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).

[0093] Each execution unit in execution unit array 1208A-1208N 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 Arithmetic Logic Units (ALUs) or Floating Point Units (FPUs) for a particular graphics processor. In some embodiments, execution units 1208A-1208N support integer and floating-point data types.

[0094] 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.

[0095] One or more internal instruction caches (e.g., 1206) are included in the thread execution logic 1200 to cache thread instructions for the execution units.

In some embodiments, one or more data caches (e.g., 1212) are included to cache thread data during thread execution. In some embodiments, sampler 1210 is included to provide texture sampling for 3D operations and media sampling for media operations. In some embodiments, sampler 1210 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.

[0096] During execution, the graphics and media pipelines send thread initiation requests to thread execution logic 1200 via thread spawning and dispatch logic. In some embodiments, thread execution logic 1200 includes a local thread dispatcher 1204 that arbitrates thread initiation requests from the graphics and media pipelines and instantiates the requested threads on one or more execution units 1208A-1208N. For example, the geometry pipeline (e.g., 1136 of Fig. 11) dispatches vertex processing, tessellation, or geometry processing threads to thread execution logic 1200 (Fig. 12). In some embodiments, thread dispatcher 1204 can also process runtime thread spawning requests from the executing shader programs.

[0097] Once a group of geometric objects has been processed and rasterized into pixel data, pixel shader 1202 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 some embodiments, pixel shader 1202 calculates the values of the various vertex attributes that are to be interpolated across the rasterized object. In some embodiments, pixel shader 1202 then executes an application programming interface (API)-supplied pixel shader program. To execute the pixel shader program, pixel shader 1202 dispatches threads to an execution unit (e.g.,

1208A) via thread dispatcher 1204. In some embodiments, pixel shader 1202 uses texture sampling logic in sampler 1210 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.

[0098] In some embodiments, the data port 1214 provides a memory access mechanism for the thread execution logic 1200 output processed data to memory for processing on a graphics processor output pipeline. In some embodiments, the data port 1214 includes or couples to one or more cache memories (e.g., data cache 1212) to cache data for memory access via the data port.

[0099] Fig. 13 is a block diagram illustrating a graphics processor instruction formats 1300 according to some embodiments. In one or more 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. In some embodiments, instruction format 1300 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.

[00100] In some embodiments, the graphics processor execution units natively support instructions in a 128-bit format 1310. A 64-bit compacted instruction format 1330 is available for some instructions based on the selected

instruction, instruction options, and number of operands. The native 128-bit format 1310 provides access to all instruction options, while some options and operations are restricted in the 64-bit format 1330. The native instructions available in the 64-bit format 1330 vary by embodiment. In some embodiments, the instruction is compacted in part using a set of index values in an index field 1313. 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 1310.

[00101] For each format, instruction opcode 1312 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. In some embodiments, instruction control field 1314 enables control over certain execution options, such as channels selection (e.g., predication) and data channel order (e.g., swizzle). For 128-bit instructions 1310 an exec-size field 1316 limits the number of data channels that will be executed in parallel. In some embodiments, exec-size field 1316 is not available for use in the 64-bit compact instruction format 1330.

[00102] Some execution unit instructions have up to three operands including two source operands, src0 1322, src1 1322, and one destination 1318. In some embodiments, 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 1324), where the instruction opcode 1312 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.

[00103] In some embodiments, the 128-bit instruction format 1310 includes an access/address mode information 1326 specifying, for example, whether direct register addressing mode or indirect register addressing mode is used. When direct register addressing mode is used, the register address of one or more operands is directly provided by bits in the instruction 1310.

[00104] In some embodiments, the 128-bit instruction format 1310 includes an access/address mode field 1326, which specifies an address mode and/or an access mode for the instruction. In one embodiment the access mode to define a data access alignment for the instruction. Some embodiments support access modes including a 16-byte aligned access mode and a 1-byte aligned access mode, where the byte alignment of the access mode determines the access alignment of the instruction operands. For example, when in a first mode, the instruction 1310 may use byte-aligned addressing for source and destination operands and when in a second mode, the instruction 1310 may use 16-byte-aligned addressing for all source and destination operands.

[00105] In one embodiment, the address mode portion of the access/address mode field 1326 determines whether the instruction is to use direct or indirect addressing. When direct register addressing mode is used bits in the instruction 1310 directly provide the register address of one or more operands. When indirect

register addressing mode is used, the register address of one or more operands may be computed based on an address register value and an address immediate field in the instruction.

[00106] In some embodiments instructions are grouped based on opcode 1312 bit-fields to simplify Opcode decode 1340. For an 8-bit opcode, bits 10, 11, and 12 allow the execution unit to determine the type of opcode. The precise opcode grouping shown is merely an example. In some embodiments, a move and logic opcode group 1342 includes data movement and logic instructions (e.g., move (mov), compare (cmp)). In some embodiments, move and logic group 1342 shares the five most significant bits (MSB), where move (mov) instructions are in the form of 0000xxxxb and logic instructions are in the form of 0001xxxxb. A flow control instruction group 1344 (e.g., call, jump (jmp)) includes instructions in the form of 0010xxxxb (e.g., 0x20). A miscellaneous instruction group 1346 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 1348 includes component-wise arithmetic instructions (e.g., add, multiply (mul)) in the form of 0100xxxxb (e.g., 0x40). The parallel math group 1348 performs the arithmetic operations in parallel across data channels. The vector math group 1350 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.

[00107] Fig. 14 is a block diagram of another embodiment of a graphics processor 1400. Elements of Fig. 14 having the same reference numbers (or names)

as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

[00108] In some embodiments, graphics processor 1400 includes a graphics pipeline 1420, a media pipeline 1430, a display engine 1440, thread execution logic 1450, and a render output pipeline 1470. In some embodiments, graphics processor 1400 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 graphics processor 1400 via a ring interconnect 1402. In some embodiments, ring interconnect 1402 couples graphics processor 1400 to other processing components, such as other graphics processors or general-purpose processors. Commands from ring interconnect 1402 are interpreted by a command streamer 1403, which supplies instructions to individual components of graphics pipeline 1420 or media pipeline 1430.

[00109] In some embodiments, command streamer 1403 directs the operation of a vertex fetcher 1405 that reads vertex data from memory and executes vertex-processing commands provided by command streamer 1403. In some embodiments, vertex fetcher 1405 provides vertex data to a vertex shader 1407, which performs coordinate space transformation and lighting operations to each vertex. In some embodiments, vertex fetcher 1405 and vertex shader 1407 execute vertex-processing instructions by dispatching execution threads to execution units 1452A, 1452B via a thread dispatcher 1431.

[00110] In some embodiments, execution units 1452A, 1452B are an array of vector processors having an instruction set for performing graphics and media operations. In some embodiments, execution units 1452A, 1452B have an attached L1 cache 1451 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.

[00111] In some embodiments, graphics pipeline 1420 includes tessellation components to perform hardware-accelerated tessellation of 3D objects. In some embodiments, a programmable hull shader 1411 configures the tessellation operations. A programmable domain shader 1417 provides back-end evaluation of tessellation output. A tessellator 1413 operates at the direction of hull shader 1411 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 graphics pipeline 1420. In some embodiments, if tessellation is not used, tessellation components 1411, 1413, 1417 can be bypassed.

[00112] In some embodiments, complete geometric objects can be processed by a geometry shader 1419 via one or more threads dispatched to execution units 1452A, 1452B, or can proceed directly to the clipper 1429. In some embodiments, 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 1419 receives input from the vertex shader 1407. In some embodiments, geometry shader 1419 is programmable by a geometry

shader program to perform geometry tessellation if the tessellation units are disabled.

[00113] Before rasterization, a clipper 1429 processes vertex data. The clipper 1429 may be a fixed function clipper or a programmable clipper having clipping and geometry shader functions. In some embodiments, a rasterizer/depth 1473 in the render output pipeline 1470 dispatches pixel shaders to convert the geometric objects into their per pixel representations. In some embodiments, pixel shader logic is included in thread execution logic 1450. In some embodiments, an application can bypass the rasterizer 1473 and access un-rasterized vertex data via a stream out unit 1423.

[00114] The graphics processor 1400 has an interconnect bus, interconnect fabric, or some other interconnect mechanism that allows data and message passing amongst the major components of the processor. In some embodiments, execution units 1452A, 1452B and associated cache(s) 1451, texture and media sampler 1454, and texture/sampler cache 1458 interconnect via a data port 1456 to perform memory access and communicate with render output pipeline components of the processor. In some embodiments, sampler 1454, caches 1451, 1458 and execution units 1452A, 1452B each have separate memory access paths.

[00115] In some embodiments, render output pipeline 1470 contains a rasterizer and depth test component 1473 that converts vertex-based objects into an associated pixel-based representation. In some embodiments, the rasterizer logic includes a windower/masker unit to perform fixed function triangle and line rasterization. An associated render cache 1478 and depth cache 1479 are also

available in some embodiments. A pixel operations component 1477 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 1441, or substituted at display time by the display controller 1443 using overlay display planes. In some embodiments, a shared L3 cache 1475 is available to all graphics components, allowing the sharing of data without the use of main system memory.

[00116] In some embodiments, graphics processor media pipeline 1430 includes a media engine 1437 and a video front end 1434. In some embodiments, video front end 1434 receives pipeline commands from the command streamer 1403. In some embodiments, media pipeline 1430 includes a separate command streamer. In some embodiments, video front-end 1434 processes media commands before sending the command to the media engine 1437. In some embodiments, media engine 1437 includes thread spawning functionality to spawn threads for dispatch to thread execution logic 1450 via thread dispatcher 1431.

[00117] In some embodiments, graphics processor 1400 includes a display engine 1440. In some embodiments, display engine 1440 is external to processor 1400 and couples with the graphics processor via the ring interconnect 1402, or some other interconnect bus or fabric. In some embodiments, display engine 1440 includes a 2D engine 1441 and a display controller 1443. In some embodiments, display engine 1440 contains special purpose logic capable of operating independently of the 3D pipeline. In some embodiments, display controller 1443 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 a display device connector.

[00118] In some embodiments, graphics pipeline 1420 and media pipeline 1430 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 some embodiments, 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 some embodiments, support is provided for the Open Graphics Library (OpenGL) and Open Computing Language (OpenCL) from the Khronos Group, the Direct3D library from the Microsoft Corporation, or support may be provided to 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.

[00119] Fig. 15A is a block diagram illustrating a graphics processor command format 1500 according to some embodiments. Fig. 15B is a block diagram illustrating a graphics processor command sequence 1510 according to an embodiment. The solid lined boxes in Fig. 15A 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 1500 of Fig. 15A includes data fields to identify a target client 1502 of the command, a command

operation code (opcode) 1504, and the relevant data 1506 for the command. A sub-opcode 1505 and a command size 1508 are also included in some commands.

[00120] In some embodiments, client 1502 specifies the client unit of the graphics device that processes the command data. In some embodiments, 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 some embodiments, 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 1504 and, if present, sub-opcode 1505 to determine the operation to perform. The client unit performs the command using information in data field 1506. For some commands an explicit command size 1508 is expected to specify the size of the command. In some embodiments, the command parser automatically determines the size of at least some of the commands based on the command opcode. In some embodiments commands are aligned via multiples of a double word.

[00121] The flow diagram in Fig. 15B shows an exemplary graphics processor command sequence 1510. In some embodiments, software or firmware of a data processing system that features an embodiment of a 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 purposes of example only as embodiments are not limited to these specific

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 at least partially concurrence.

[00122] In some embodiments, the graphics processor command sequence 1510 may begin with a pipeline flush command 1512 to cause any active graphics pipeline to complete the currently pending commands for the pipeline. In some embodiments, the 3D pipeline 1522 and the media pipeline 1524 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. In some embodiments, pipeline flush command 1512 can be used for pipeline synchronization or before placing the graphics processor into a low power state.

[00123] In some embodiments, a pipeline select command 1513 is used when a command sequence requires the graphics processor to explicitly switch between pipelines. In some embodiments, a pipeline select command 1513 is required only once within an execution context before issuing pipeline commands unless the context is to issue commands for both pipelines. In some embodiments, a pipeline flush command is 1512 is required immediately before a pipeline switch via the pipeline select command 1513.

[00124] In some embodiments, a pipeline control command 1514 configures a graphics pipeline for operation and is used to program the 3D pipeline 1522 and the media pipeline 1524. In some embodiments, pipeline control command 1514 configures the pipeline state for the active pipeline. In one embodiment, the pipeline control command 1514 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.

[00125] In some embodiments, return buffer state commands 1516 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. In some embodiments, the graphics processor also uses one or more return buffers to store output data and to perform cross thread communication. In some embodiments, the return buffer state 1516 includes selecting the size and number of return buffers to use for a set of pipeline operations.

[00126] The remaining commands in the command sequence differ based on the active pipeline for operations. Based on a pipeline determination 1520, the command sequence is tailored to the 3D pipeline 1522 beginning with the 3D pipeline state 1530, or the media pipeline 1524 beginning at the media pipeline state 1540.

[00127] The commands for the 3D pipeline state 1530 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. In some embodiments, 3D pipeline state 1530 commands are also able to selectively disable or bypass certain pipeline elements if those elements will not be used.

[00128] In some embodiments, 3D primitive 1532 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 1532 command are forwarded to the vertex fetch function in the graphics pipeline. The vertex fetch function uses the 3D primitive 1532 command data to generate vertex data structures. The vertex data structures are stored in one or more return buffers. In some embodiments, 3D primitive 1532 command is used to perform vertex operations on 3D primitives via vertex shaders. To process vertex shaders, 3D pipeline 1522 dispatches shader execution threads to graphics processor execution units.

[00129] In some embodiments, 3D pipeline 1522 is triggered via an execute 1534 command or event. In some embodiments, a register write triggers command execution. In some embodiments 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.

[00130] In some embodiments, the graphics processor command sequence 1510 follows the media pipeline 1524 path when performing media operations. In general, the specific use and manner of programming for the media pipeline 1524 depends on the media or compute operations to be performed. Specific media decode operations may be offloaded to the media pipeline during media decode. In some embodiments, 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.

[00131] In some embodiments, media pipeline 1524 is configured in a similar manner as the 3D pipeline 1522. A set of media pipeline state commands 1540 are dispatched or placed into a command queue before the media object commands 1542. In some embodiments, media pipeline state commands 1540 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. In some embodiments, media pipeline state commands 1540 also support the use one or more pointers to “indirect” state elements that contain a batch of state settings.

[00132] In some embodiments, media object commands 1542 supply pointers to media objects for processing by the media pipeline. The media objects include memory buffers containing video data to be processed. In some embodiments, all media pipeline states must be valid before issuing a media object command 1542. Once the pipeline state is configured and media object commands 1542 are queued, the media pipeline 1524 is triggered via an execute command 1544 or an equivalent execute event (e.g., register write). Output from media pipeline 1524 may then be post processed by operations provided by the 3D pipeline 1522 or the media pipeline 1524. In some embodiments, GPGPU operations are configured and executed in a similar manner as media operations.

[00133] Fig. 16 illustrates exemplary graphics software architecture for a data processing system 1600 according to some embodiments. In some embodiments, software architecture includes a 3D graphics application 1610, an operating system 1620, and at least one processor 1630. In some embodiments, processor 1630 includes a graphics processor 1632 and one or more general-purpose processor core(s) 1634. The graphics application 1610 and operating system 1620 each execute in the system memory 1650 of the data processing system.

[00134] In some embodiments, 3D graphics application 1610 contains one or more shader programs including shader instructions 1612. 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 1614 in a machine language suitable for execution

by the general-purpose processor core 1634. The application also includes graphics objects 1616 defined by vertex data.

[00135] In some embodiments, operating system 1620 is 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 1620 uses a front-end shader compiler 1624 to compile any shader instructions 1612 in HLSL into a lower-level shader language. The compilation may be a just-in-time (JIT) compilation or the application can perform shader pre-compilation. In some embodiments, high-level shaders are compiled into low-level shaders during the compilation of the 3D graphics application 1610.

[00136] In some embodiments, user mode graphics driver 1626 contains a back-end shader compiler 1627 to convert the shader instructions 1612 into a hardware specific representation. When the OpenGL API is in use, shader instructions 1612 in the GLSL high-level language are passed to a user mode graphics driver 1626 for compilation. In some embodiments, user mode graphics driver 1626 uses operating system kernel mode functions 1628 to communicate with a kernel mode graphics driver 1629. In some embodiments, kernel mode graphics driver 1629 communicates with graphics processor 1632 to dispatch commands and instructions.

[00137] One or more aspects of at least one embodiment may be implemented by representative code stored on a machine-readable medium which represents and/or defines logic within an integrated circuit such as a processor. For

example, the machine-readable medium may include instructions which represent various logic within the processor. When read by a machine, the instructions may cause the machine to fabricate the logic to perform the techniques described herein. Such representations, known as “IP cores,” are reusable units of logic for an integrated circuit that may be stored on a tangible, machine-readable medium as a hardware model that describes the structure of the integrated circuit. The hardware model may be supplied to various customers or manufacturing facilities, which load the hardware model on fabrication machines that manufacture the integrated circuit. The integrated circuit may be fabricated such that the circuit performs operations described in association with any of the embodiments described herein.

[00138] Fig. 17 is a block diagram illustrating an IP core development system 1700 that may be used to manufacture an integrated circuit to perform operations according to an embodiment. The IP core development system 1700 may be used to generate modular, re-usable designs that can be incorporated into a larger design or used to construct an entire integrated circuit (e.g., an SOC integrated circuit). A design facility 1730 can generate a software simulation 1710 of an IP core design in a high level programming language (e.g., C/C++). The software simulation 1710 can be used to design, test, and verify the behavior of the IP core. A register transfer level (RTL) design can then be created or synthesized from the simulation model 1700. The RTL design 1715 is an abstraction of the behavior of the integrated circuit that models the flow of digital signals between hardware registers, including the associated logic performed using the modeled digital signals. In addition to an RTL design 1715, lower-level designs at the logic level or transistor

level may also be created, designed, or synthesized. Thus, the particular details of the initial design and simulation may vary.

[00139] The RTL design 1715 or equivalent may be further synthesized by the design facility into a hardware model 1720, which may be in a hardware description language (HDL), or some other representation of physical design data. The HDL may be further simulated or tested to verify the IP core design. The IP core design can be stored for delivery to a 3rd party fabrication facility 1765 using non-volatile memory 1740 (e.g., hard disk, flash memory, or any non-volatile storage medium). Alternatively, the IP core design may be transmitted (e.g., via the Internet) over a wired connection 1750 or wireless connection 1760. The fabrication facility 1765 may then fabricate an integrated circuit that is based at least in part on the IP core design. The fabricated integrated circuit can be configured to perform operations in accordance with at least one embodiment described herein.

[00140] Fig. 18 is a block diagram illustrating an exemplary system on a chip integrated circuit 1800 that may be fabricated using one or more IP cores, according to an embodiment. The exemplary integrated circuit includes one or more application processors 1805 (e.g., CPUs), at least one graphics processor 1810, and may additionally include an image processor 1815 and/or a video processor 1820, any of which may be a modular IP core from the same or multiple different design facilities. The integrated circuit includes peripheral or bus logic including a USB controller 1825, UART controller 1830, an SPI/SDIO controller 1835, and an I2S/I2C controller 1840. Additionally, the integrated circuit can

include a display device 1845 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1850 and a mobile industry processor interface (MIPI) display interface 1855. Storage may be provided by a flash memory subsystem 1860 including flash memory and a flash memory controller. Memory interface may be provided via a memory controller 1865 for access to SDRAM or SRAM memory devices. Some integrated circuits additionally include an embedded security engine 1870.

[00141] Additionally, other logic and circuits may be included in the processor of integrated circuit 1800, including additional graphics processors/cores, peripheral interface controllers, or general purpose processor cores.

[00142] The following examples pertain to further embodiments. Example 1 includes an apparatus comprising: Environmental Mapping (EM) logic to perform one or more operations to extract illumination information for an object from an environmental map in response to a determination that the object includes a diffuse surface; and memory, coupled to the EM logic, to store data corresponding to the environmental map. Example 2 includes the apparatus of example 1, wherein the EM logic is to cause a Graphics Processing Unit (GPU) to perform the one or more operations in response to the determination that the object has the diffuse surface. Example 3 includes the apparatus of any one of examples 1-2, wherein the EM logic is to prioritize processing of pixels on contours or boundaries of the object over other pixels of the object. Example 4 includes the apparatus of any one of examples 1-3, comprising update logic to update data corresponding to the object, or a scene that is to comprise the object, in response to user input. Example 5

includes the apparatus of any one of examples 1-4, comprising update logic to update data corresponding to the object, or a scene that is to comprise the object, in response to user input and after the GPU has rendered the object or the scene. Example 6 includes the apparatus of any one of examples 1-5, wherein the user input is to comprise customization of three dimensional virtual objects with three dimensional scanning from real world or from models generated on a computing device. Example 7 includes the apparatus of any one of examples 1-6, wherein the EM logic is to perform the one or more operations based at least in part on a weighed sum of pixels in the environmental map. Example 8 includes the apparatus of any one of examples 1-7, wherein the EM logic is to perform the one or more operations based at least in part on at least nine coefficients corresponding to lowest-frequency modes of illumination. Example 9 includes the apparatus of any one of examples 1-8, wherein the EM logic is to treat each color channel separately to allow the coefficients to operate as separate RGB (Red, Green, Blue) values. Example 10 includes the apparatus of any one of examples 1-9, wherein the EM logic is to cause the GPU to render the object or a scene that is to comprise the object in response to a determination that the object includes a specular surface. Example 11 includes the apparatus of any one of examples 1-10, wherein a scene that comprises the object is to comprise: one or more three dimensional virtual objects, one or more model textures, a light distribution environment, a surrounding environment, and one or more environmental maps. Example 12 includes the apparatus of any one of examples 1-11, wherein a GPU is to comprise the EM logic. Example 13 includes the apparatus of any one of examples 1-12, wherein the object is to comprise a two dimensional object or a three dimensional

object. Example 14 includes the apparatus of any of examples 1-13, wherein a processor is to comprise the EM logic. Example 15 includes the apparatus any of examples 1-14, wherein a GPU, coupled to the EM logic, is to comprise one or more graphics processing cores. Example 16 includes the apparatus of any of examples 1-15, wherein one or more of a GPU, the EM logic, a general purpose processor core, or the memory are on a single integrated circuit die.

[00143] Example 17 includes a computing system comprising: a touch screen to receive user input; a processor, coupled to the touch screen and memory, the memory to store information corresponding to an environmental map; and Environmental Mapping (EM) logic to perform one or more operations to extract illumination information for an object from the environmental map in response to a determination that the object includes a diffuse surface. Example 18 includes the system of example 17, wherein the EM logic is to cause a Graphics Processing Unit (GPU) to perform the one or more operations in response to the determination that the object has the diffuse surface and/or specular surface. Example 19 includes the system of any one of examples 17-18, wherein the EM logic is to prioritize processing of pixels on contours or boundaries of the object over other pixels of the object. Example 20 includes the system of any one of examples 17-19, comprising update logic to update data corresponding to the object, or a scene that is to comprise the object, in response to the user input.

[00144] Example 21 includes a method comprising: performing, at EM logic, one or more operations to extract illumination information for an object from an environmental map in response to a determination that the object includes a diffuse

surface, wherein data corresponding to the environmental map is stored in memory. Example 22 includes the method of example 21, further comprising the EM logic causing a Graphics Processing Unit (GPU) to perform the one or more operations in response to the determination that the object has the diffuse surface. Example 23 includes the method of any one of examples 21-22, further comprising the EM logic prioritizing processing of pixels on contours or boundaries of the object over other pixels of the object. Example 24 includes the method of any one of examples 21-23, further comprising update logic updating data corresponding to the object, or a scene that is to comprise the object, in response to user input. Example 25 includes the method of any one of examples 21-24, further comprising update logic updating data corresponding to the object, or a scene that is to comprise the object, in response to user input and after the GPU has rendered the object or the scene. Example 26 includes the method of any one of examples 21-25, wherein the user input comprises customization of three dimensional virtual objects with three dimensional scanning from real world or from models generated on a computing device. Example 27 includes the method of any one of examples 21-26, further comprising the EM logic performing the one or more operations based at least in part on a weighed sum of pixels in the environmental map. Example 28 includes the method of any one of examples 21-27, further comprising the EM logic performing the one or more operations based at least in part on at least nine coefficients corresponding to lowest-frequency modes of illumination.

[00145] Example 29 includes a computer-readable medium comprising one or more instructions that when executed on a processor configure the processor to perform one or more operations of any one of examples 21 to 28. Example 30

includes an apparatus comprising means to perform a method as set forth in any one of examples 21 to 28.

[00146] Example 31 includes an apparatus comprising means to perform a method as set forth in any preceding example. Example 32 comprises machine-readable storage including machine-readable instructions, when executed, to implement a method or realize an apparatus as set forth in any preceding example.

[00147] In various embodiments, the operations discussed herein, e.g., with reference to Figs. 1-18, may be implemented as hardware (e.g., logic circuitry), software, firmware, or combinations thereof, which may be provided as a computer program product, e.g., including a tangible (e.g., non-transitory) machine-readable or computer-readable medium having stored thereon instructions (or software procedures) used to program a computer to perform a process discussed herein. The machine-readable medium may include a storage device such as those discussed with respect to Figs. 1-18.

[00148] Additionally, such computer-readable media may be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals provided in a carrier wave or other propagation medium via a communication link (e.g., a bus, a modem, or a network connection).

[00149] Reference in the specification to “one embodiment” or “an embodiment” means that a particular feature, structure, and/or characteristic described in connection with the embodiment may be included in at least an

implementation. The appearances of the phrase “in one embodiment” in various places in the specification may or may not be all referring to the same embodiment.

[00150] Also, in the description and claims, the terms “coupled” and “connected,” along with their derivatives, may be used. In some embodiments, “connected” may be used to indicate that two or more elements are in direct physical or electrical contact with each other. “Coupled” may mean that two or more elements are in direct physical or electrical contact. However, “coupled” may also mean that two or more elements may not be in direct contact with each other, but may still cooperate or interact with each other.

[00151] Thus, although embodiments have been described in language specific to structural features and/or methodological acts, it is to be understood that claimed subject matter may not be limited to the specific features or acts described. Rather, the specific features and acts are disclosed as sample forms of implementing the claimed subject matter.

CLAIMS

1. An apparatus comprising:

Environmental Mapping (EM) logic to perform one or more operations to extract illumination information for an object from an environmental map in response to a determination that the object includes a diffuse surface; and

memory, coupled to the EM logic, to store data corresponding to the environmental map.

2. The apparatus of claim 1, wherein the EM logic is to cause a Graphics Processing Unit (GPU) to perform the one or more operations in response to the determination that the object has the diffuse surface.

3. The apparatus of any one of claims 1-2, wherein the EM logic is to prioritize processing of pixels on contours or boundaries of the object over other pixels of the object.

4. The apparatus of any one of claims 1-3, comprising update logic to update data corresponding to the object, or a scene that is to comprise the object, in response to user input.

5. The apparatus of any one of claims 1-4, comprising update logic to update data corresponding to the object, or a scene that is to comprise the object, in response to user input and after the GPU has rendered the object or the scene.

6. The apparatus of any one of claims 1-5, wherein the user input is to comprise customization of three dimensional virtual objects with three dimensional scanning from real world or from models generated on a computing device.

7. The apparatus of any one of claims 1-6, wherein the EM logic is to perform the one or more operations based at least in part on a weighed sum of pixels in the environmental map.
8. The apparatus of any one of claims 1-7, wherein the EM logic is to perform the one or more operations based at least in part on at least nine coefficients corresponding to lowest-frequency modes of illumination.
9. The apparatus of any one of claims 1-8, wherein the EM logic is to treat each color channel separately to allow the coefficients to operate as separate RGB (Red, Green, Blue) values.
10. The apparatus of any one of claims 1-9, wherein the EM logic is to cause the GPU to render the object or a scene that is to comprise the object in response to a determination that the object includes a specular surface.
11. The apparatus of any one of claims 1-10, wherein a scene that comprises the object is to comprise: one or more three dimensional virtual objects, one or more model textures, a light distribution environment, a surrounding environment, and one or more environmental maps.
12. The apparatus of any one of claims 1-11, wherein a GPU is to comprise the EM logic.
13. The apparatus of any one of claims 1-12, wherein the object is to comprise a two dimensional object or a three dimensional object.
14. The apparatus of any of claims 1-13, wherein a processor is to comprise the EM logic.

15. The apparatus any of claims 1-14, wherein a GPU, coupled to the EM logic, is to comprise one or more graphics processing cores.

16. The apparatus of any of claims 1-15, wherein one or more of a GPU, the EM logic, a general purpose processor core, or the memory are on a single integrated circuit die.

17. A computing system comprising:

a touch screen to receive user input;

a processor, coupled to the touch screen and memory, the memory to store information corresponding to an environmental map; and

Environmental Mapping (EM) logic to perform one or more operations to extract illumination information for an object from the environmental map in response to a determination that the object includes a diffuse surface.

18. The system of claim 17, wherein the EM logic is to cause a Graphics Processing Unit (GPU) to perform the one or more operations in response to the determination that the object has the diffuse surface and/or specular surface.

19. The system of any one of claims 17-18, wherein the EM logic is to prioritize processing of pixels on contours or boundaries of the object over other pixels of the object.

20. The system of any one of claims 17-19, comprising update logic to update data corresponding to the object, or a scene that is to comprise the object, in response to the user input.

21. A method comprising:
- performing, at EM logic, one or more operations to extract illumination information for an object from an environmental map in response to a determination that the object includes a diffuse surface,
- wherein data corresponding to the environmental map is stored in memory.
22. The method of claim 21, further comprising the EM logic causing a Graphics Processing Unit (GPU) to perform the one or more operations in response to the determination that the object has the diffuse surface.
23. The method of any one of claims 21-22, further comprising the EM logic prioritizing processing of pixels on contours or boundaries of the object over other pixels of the object.
24. A computer-readable medium comprising one or more instructions that when executed on a processor configure the processor to perform one or more operations of any one of claims 21 to 23.
25. An apparatus comprising means to perform a method as set forth in any one of claims 21 to 23.

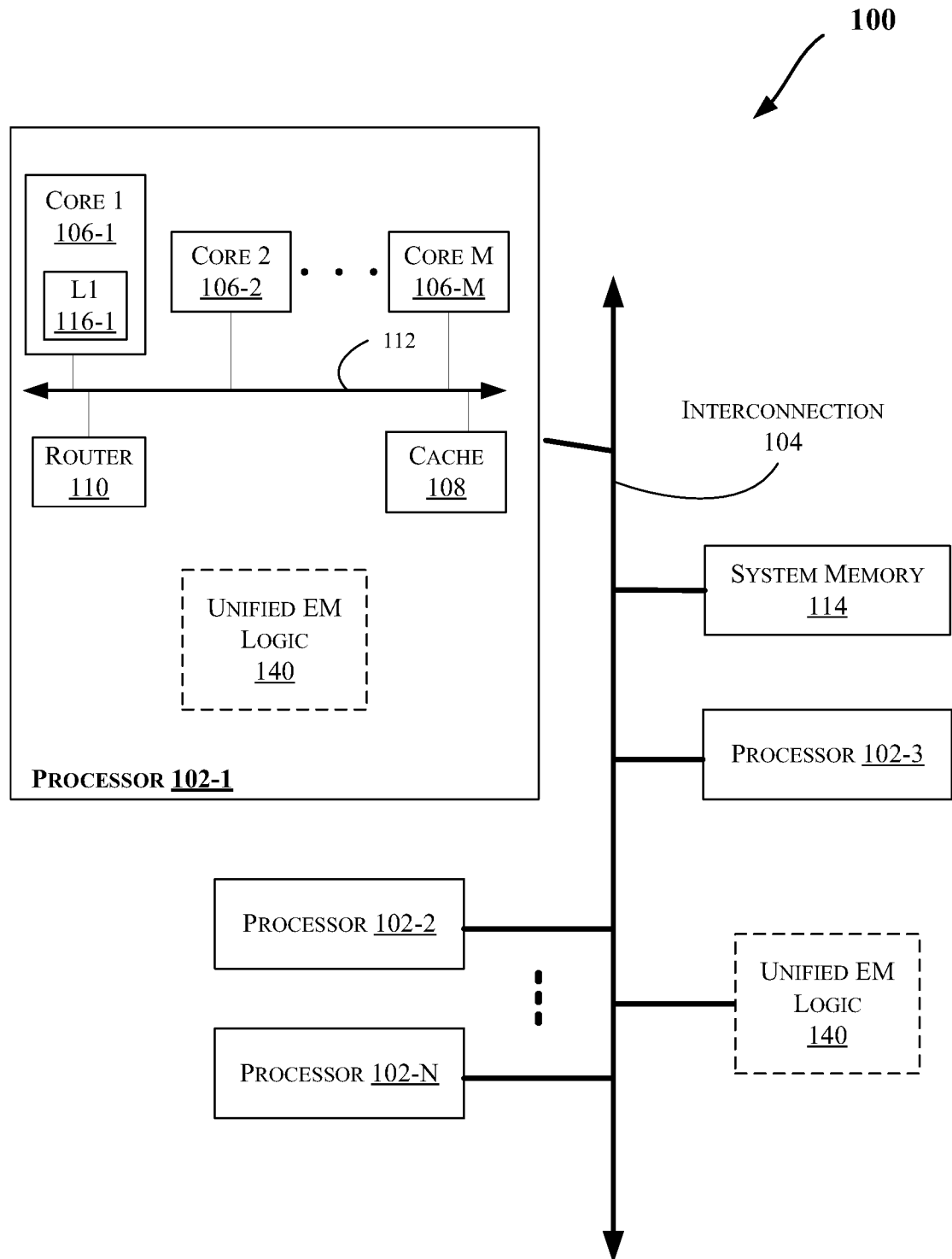
*FIG. 1*

FIG. 2A

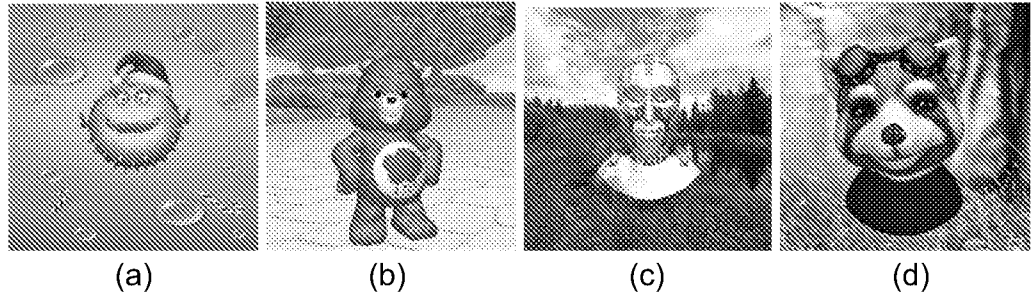


FIG. 2B

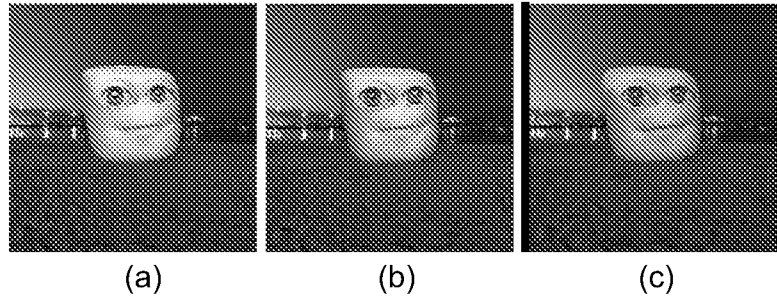


FIG. 2C

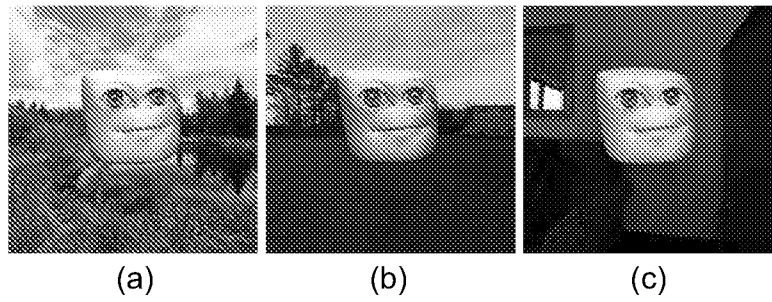


FIG. 2D

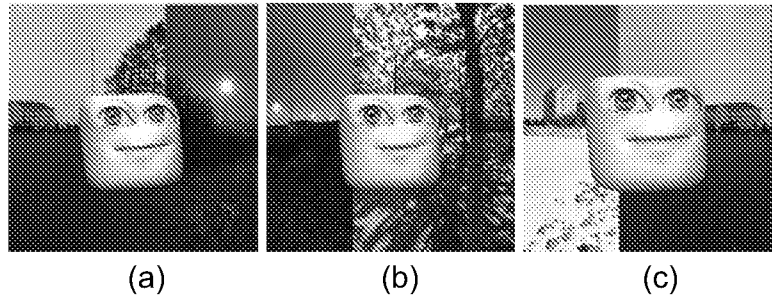
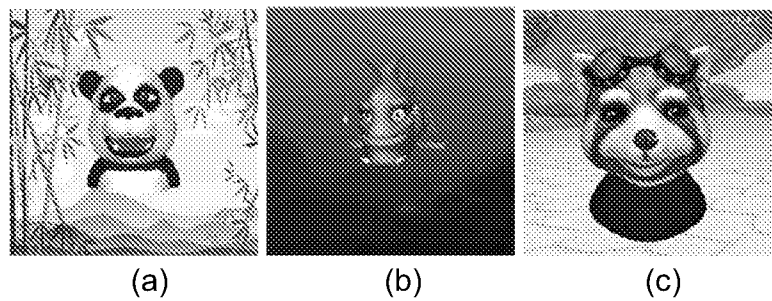


FIG. 2E



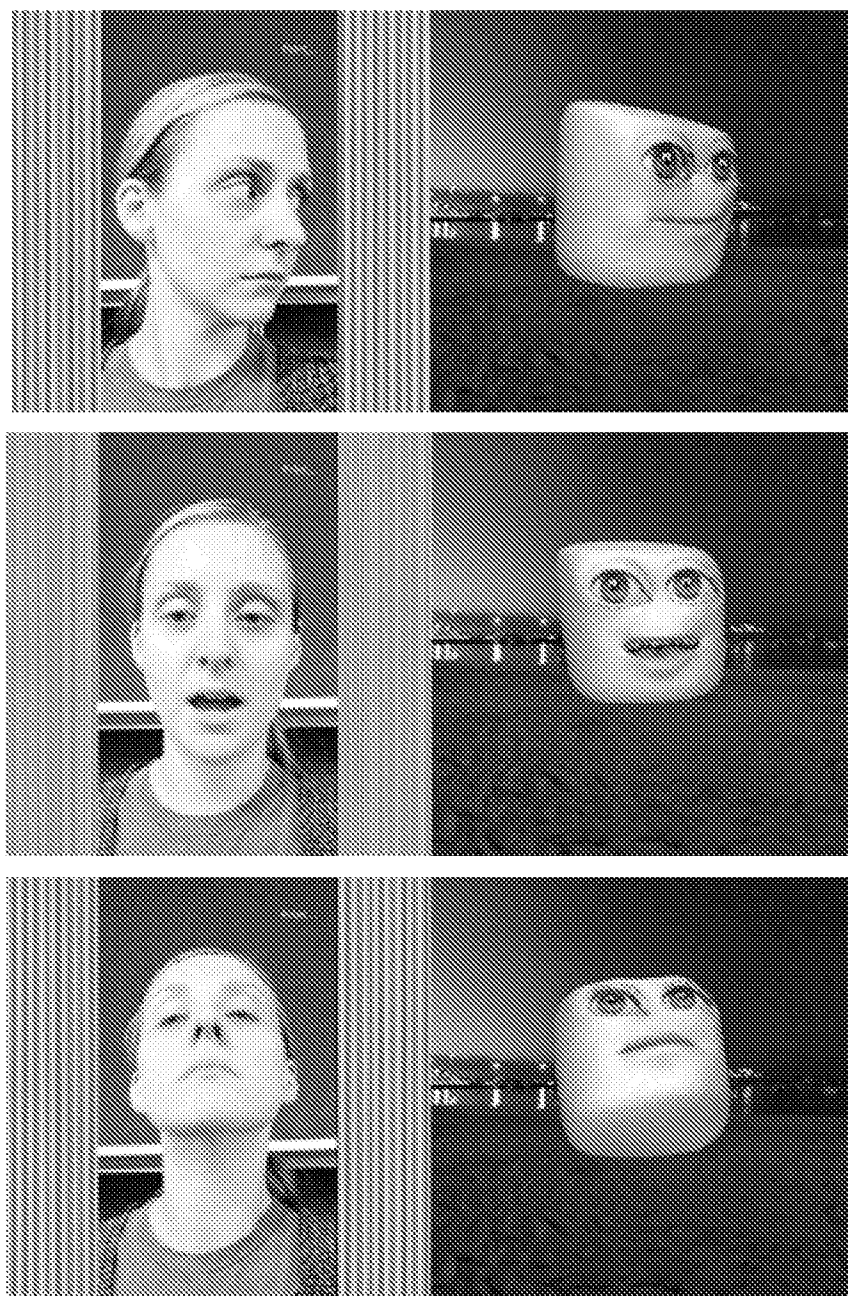


FIG. 3

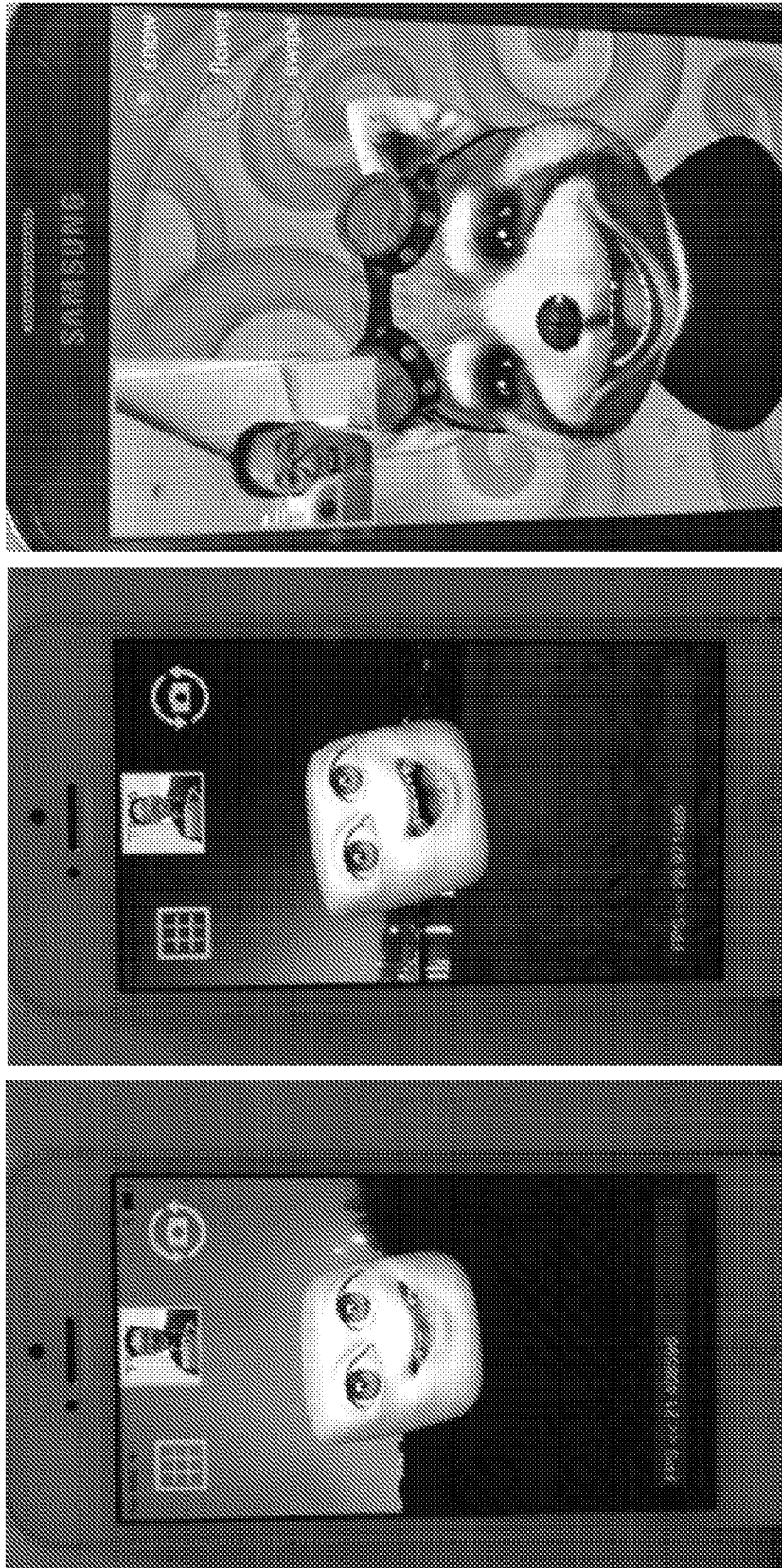
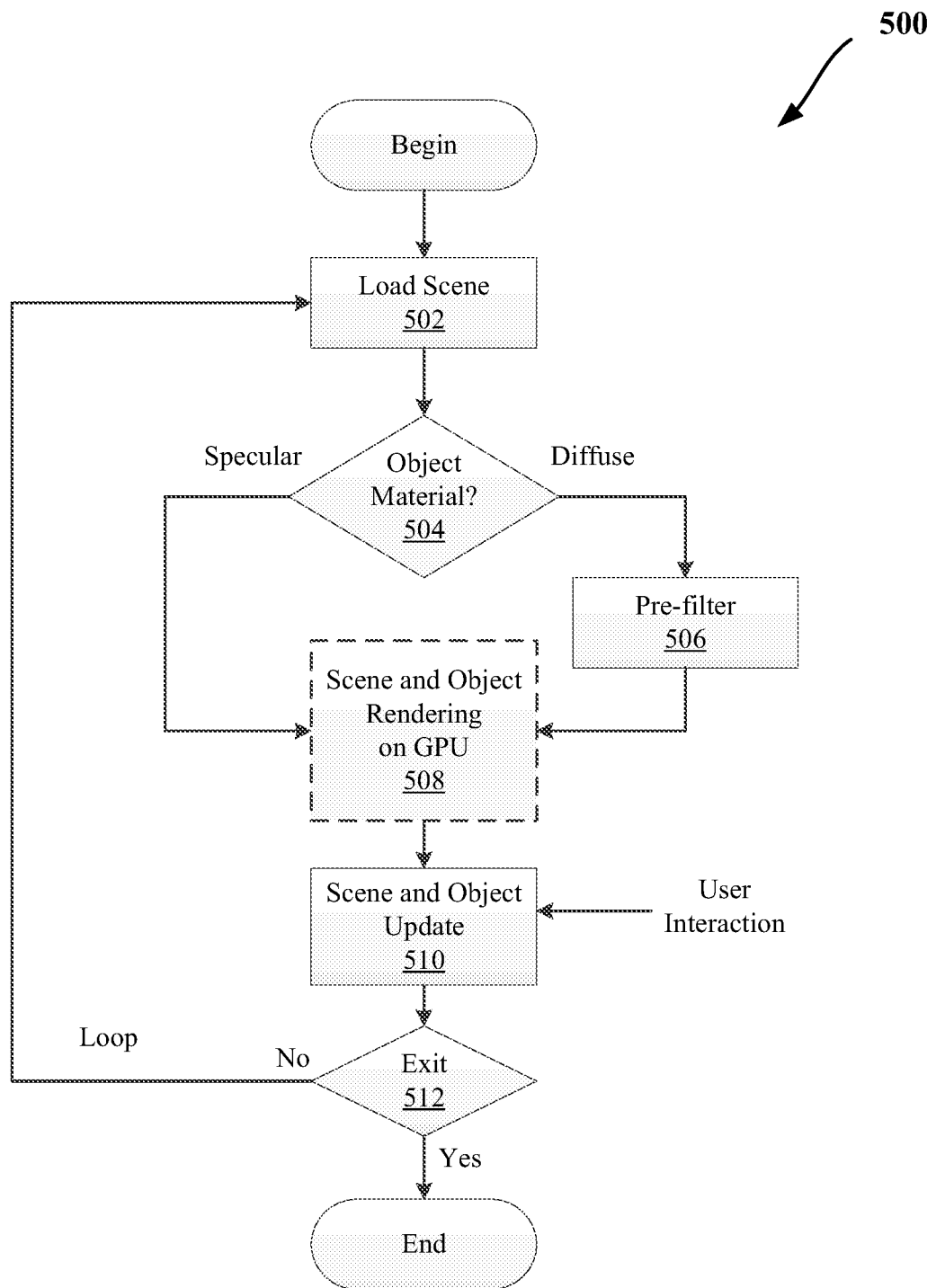


FIG. 4

*FIG. 5*

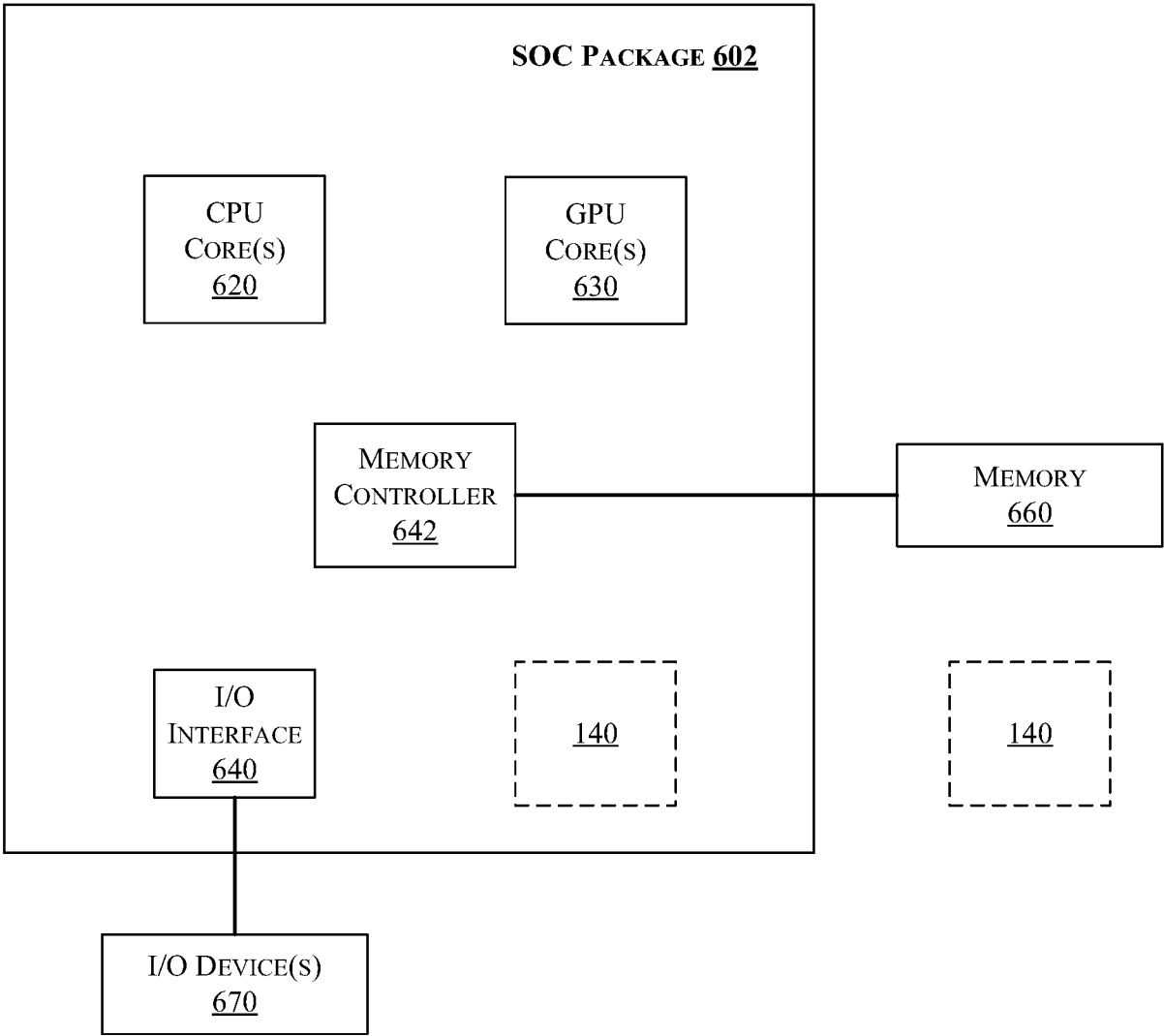
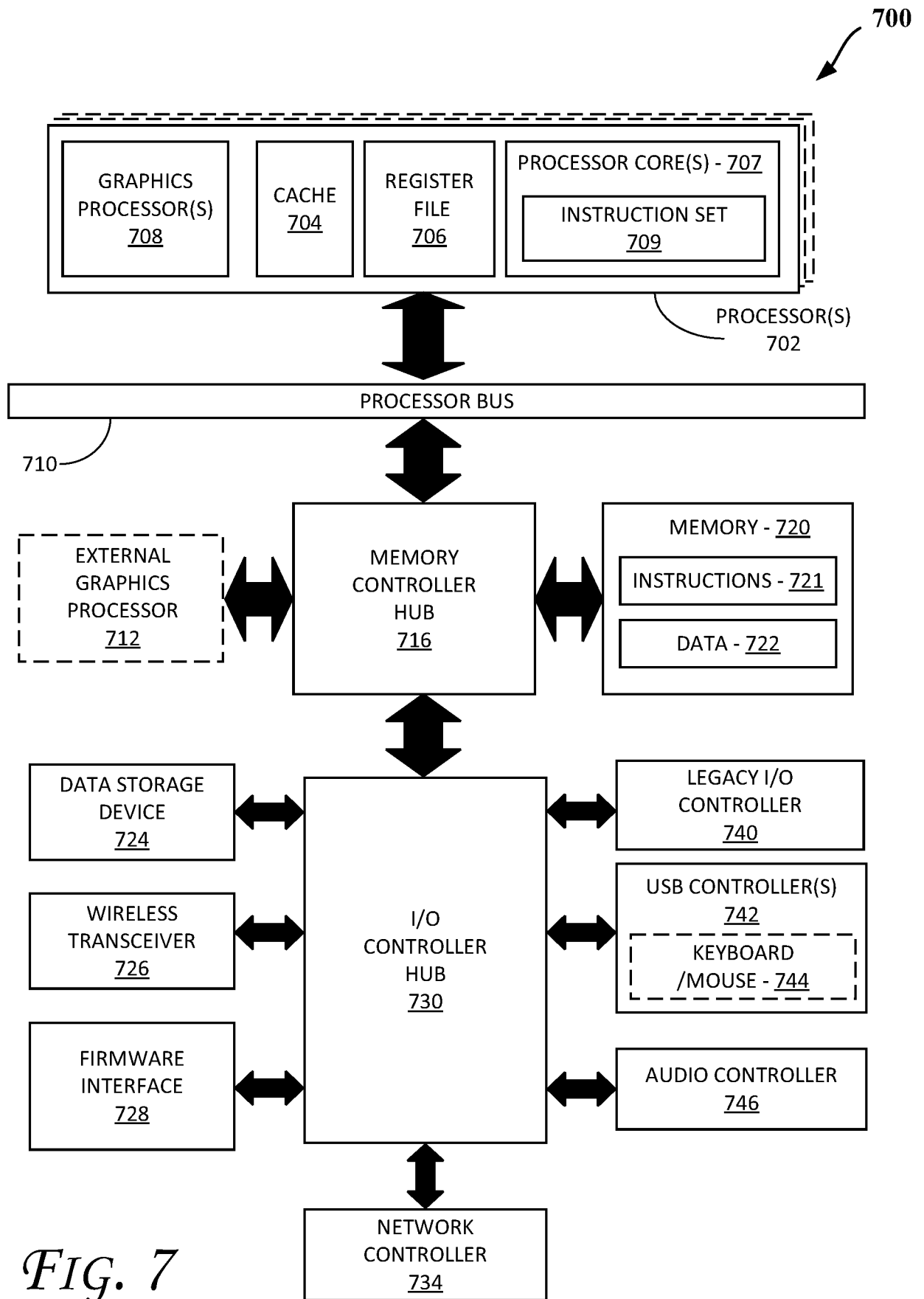


FIG. 6



PROCESSOR 800

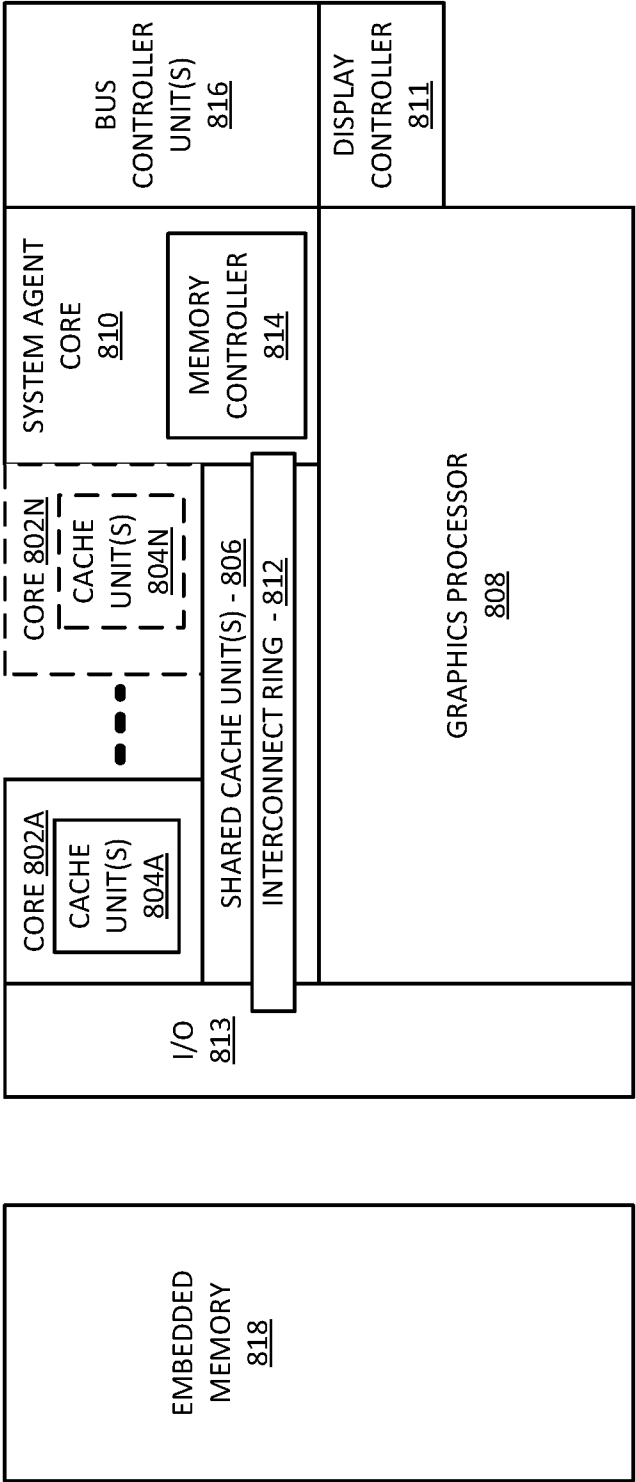


FIG. 8

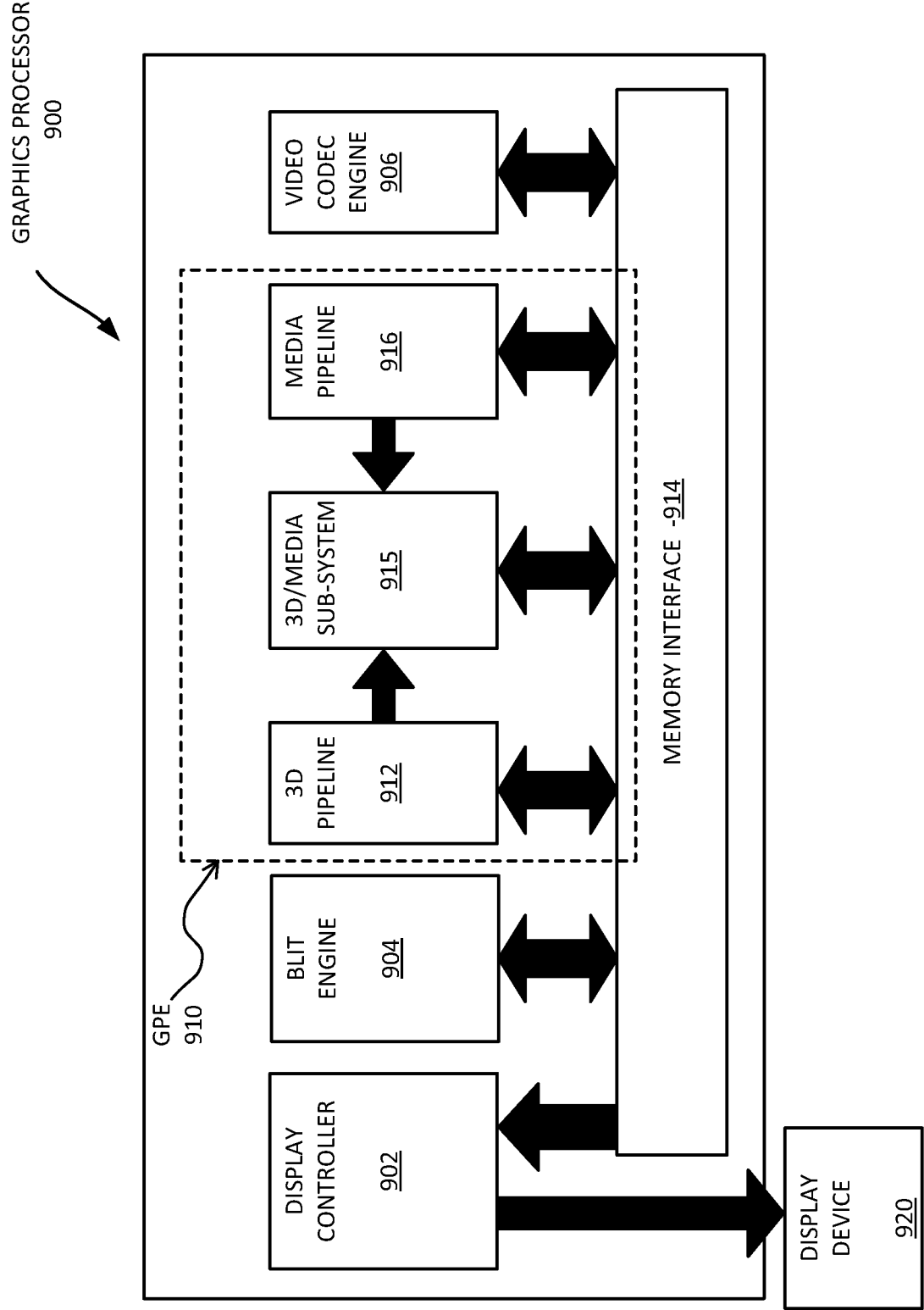


FIG. 9

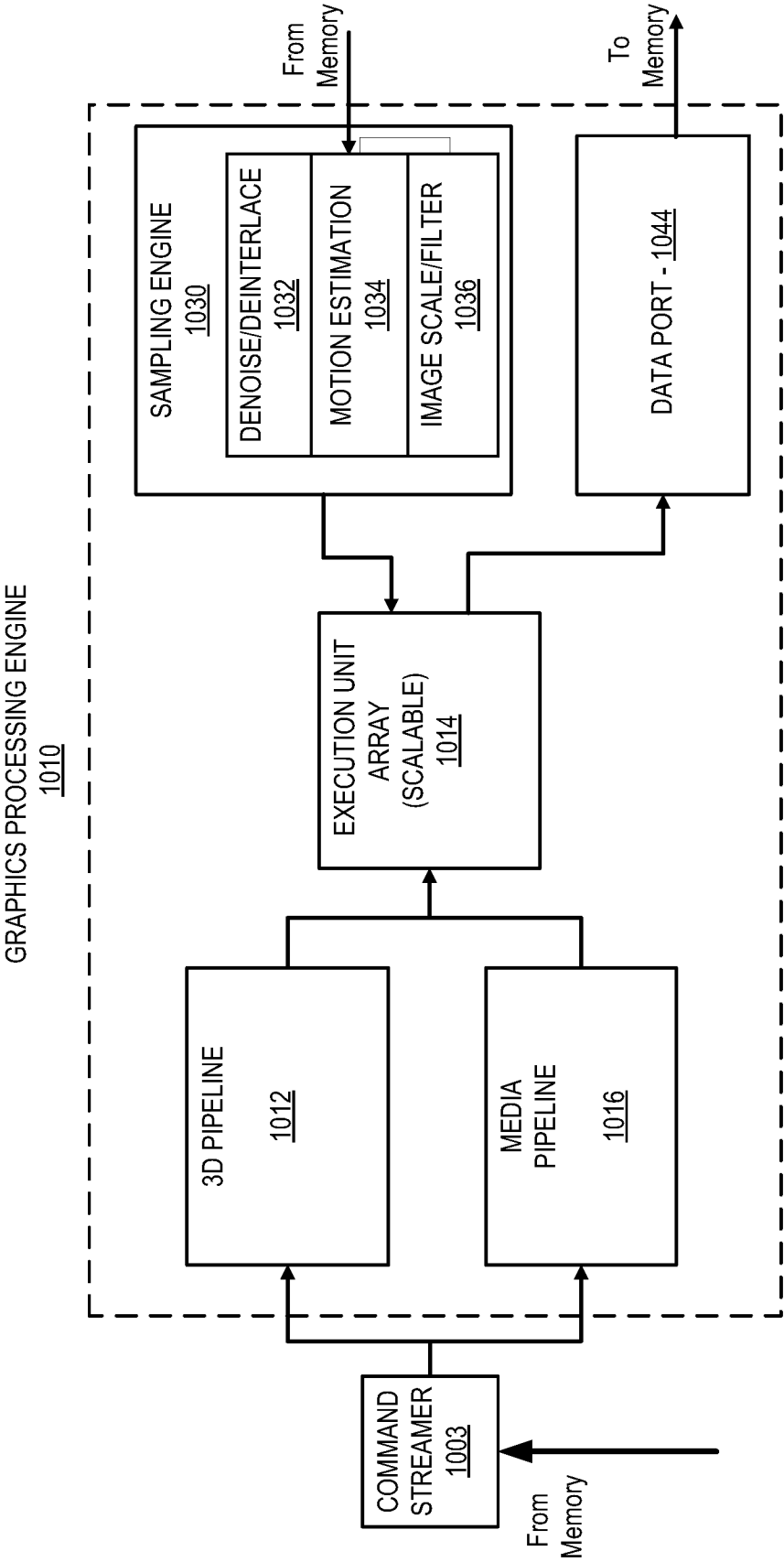


Fig. 10

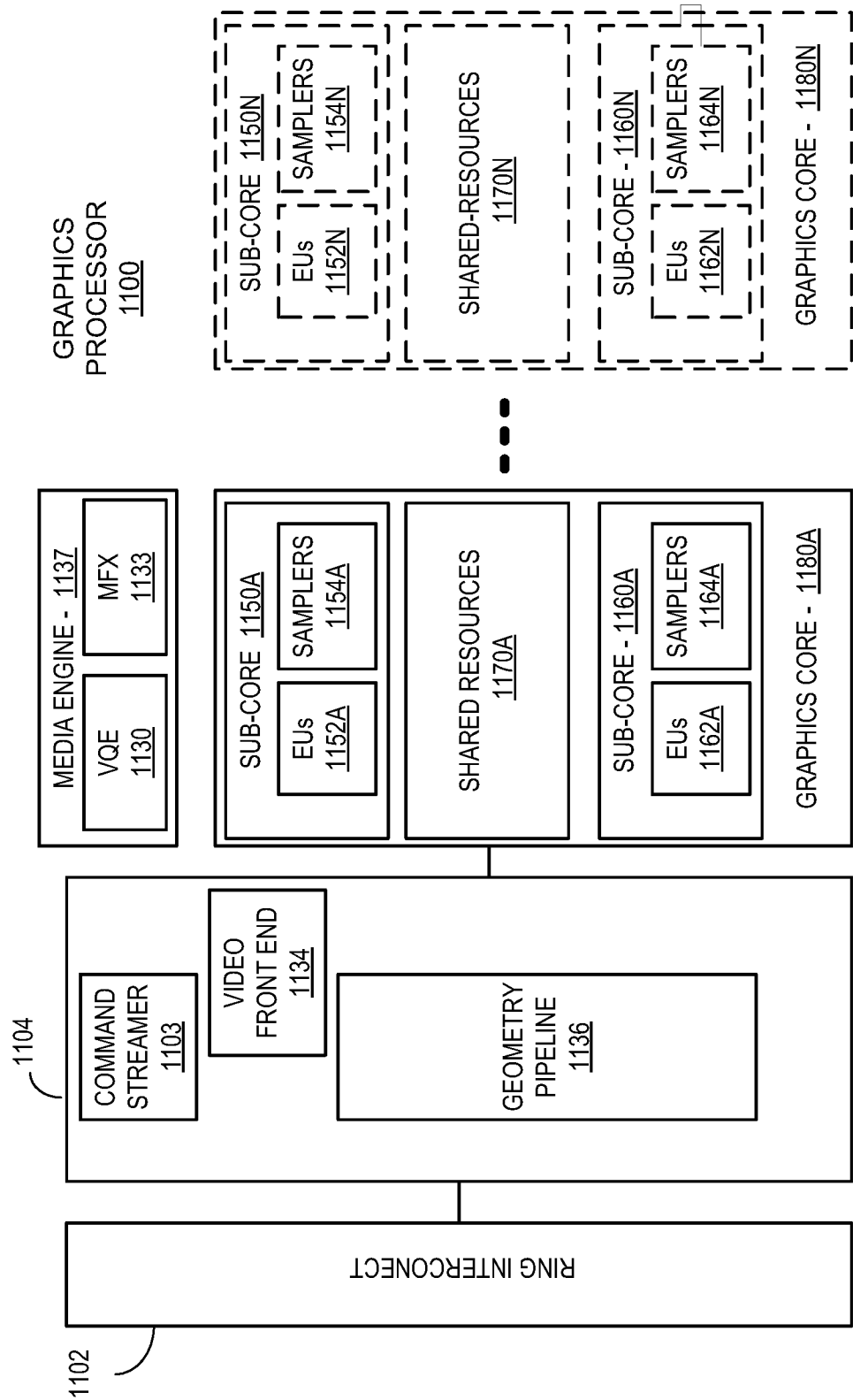


FIG. 11

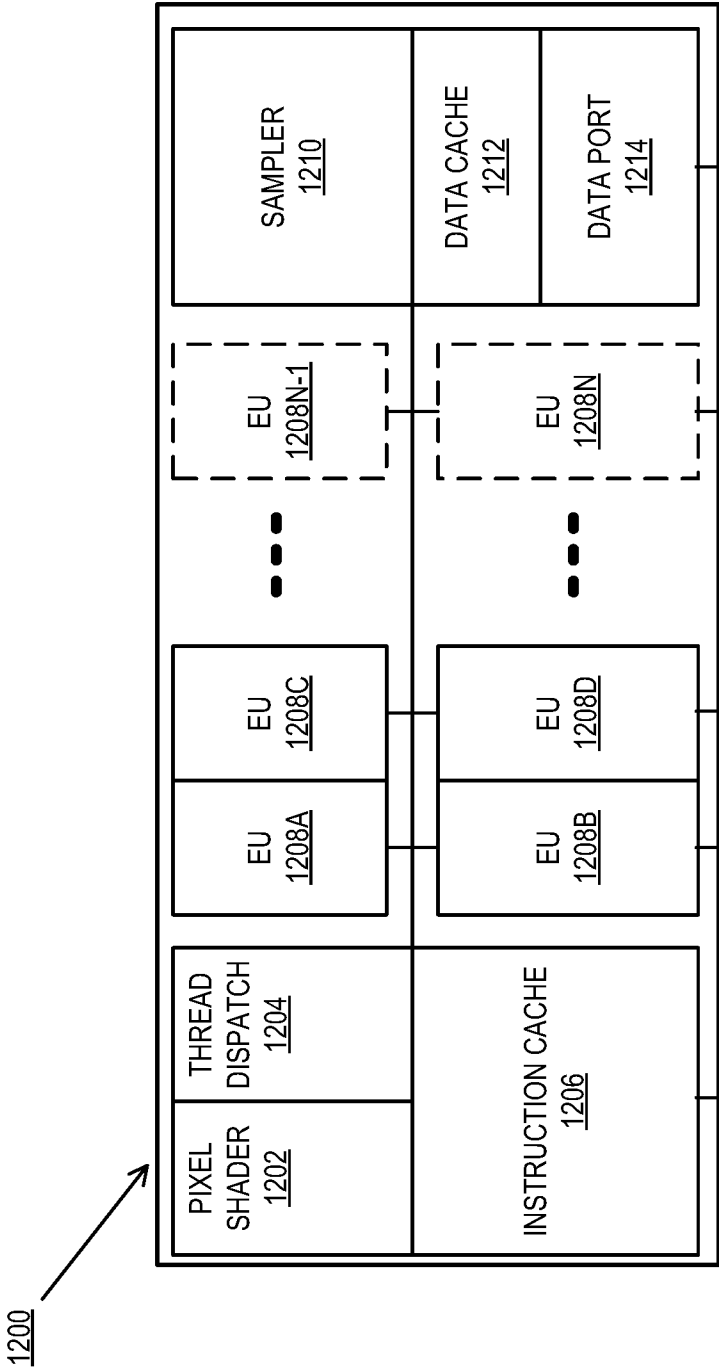


FIG. 12

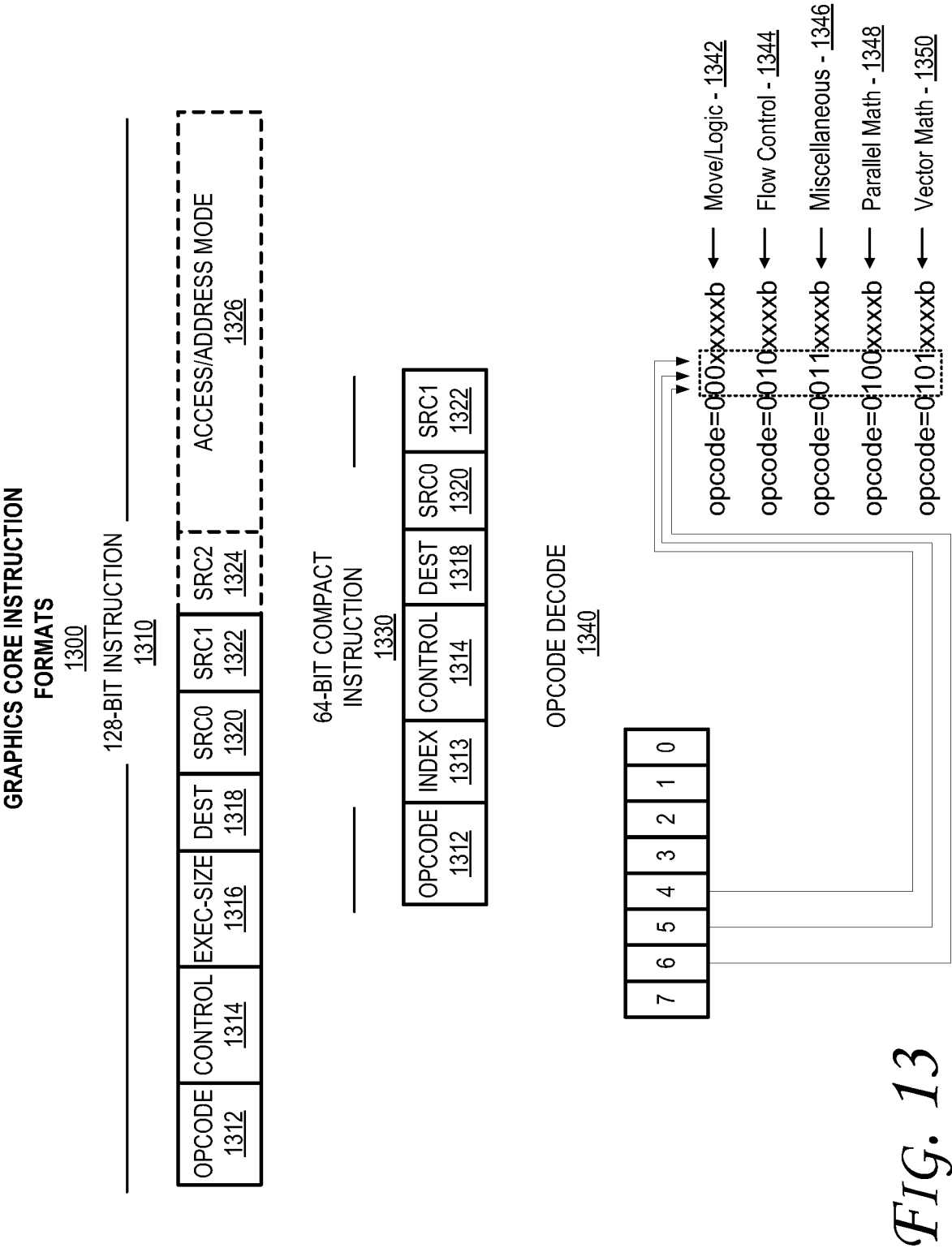


FIG. 13

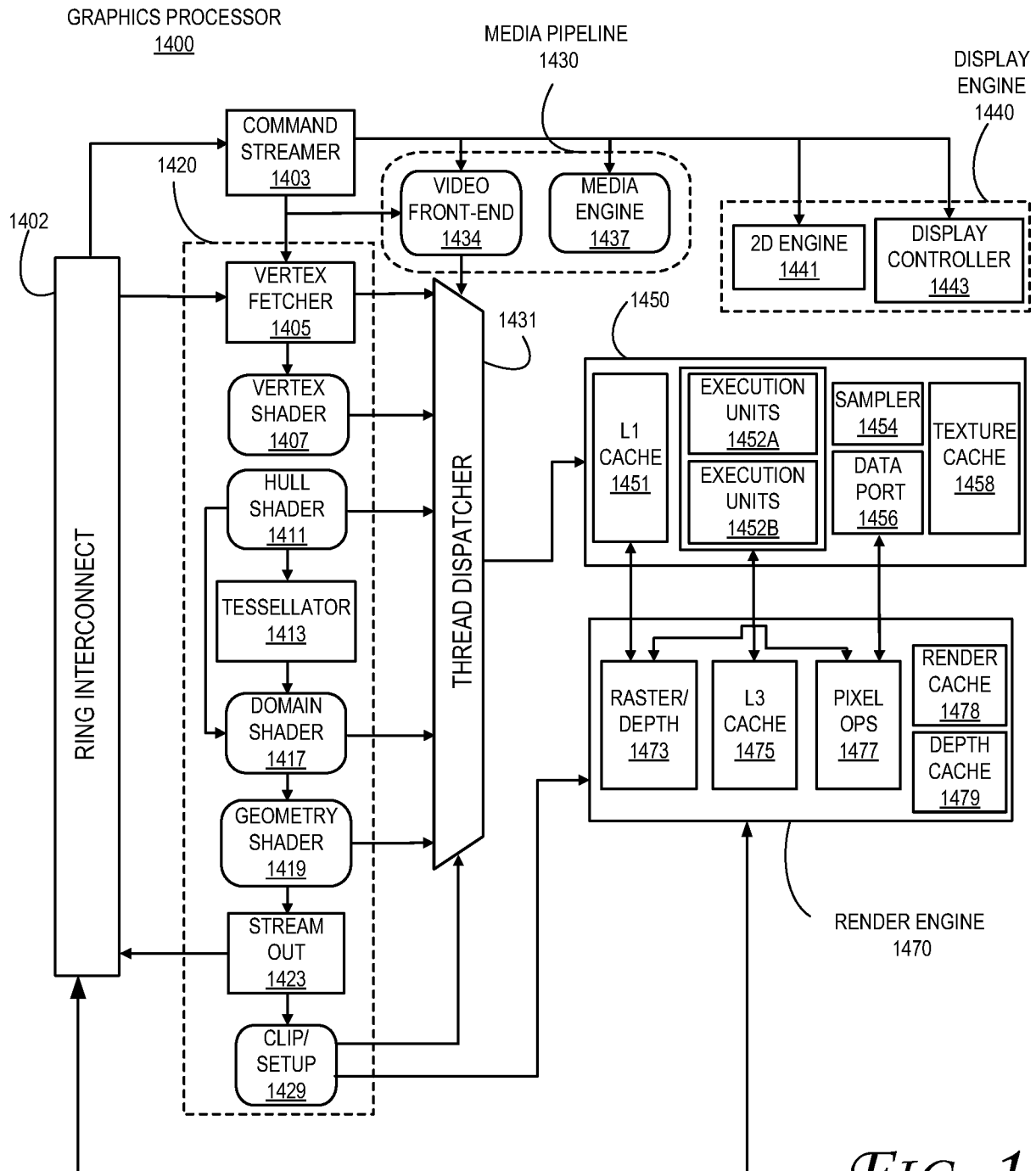


FIG. 14

FIG. 15A GRAPHICS PROCESSOR COMMAND FORMAT
1500

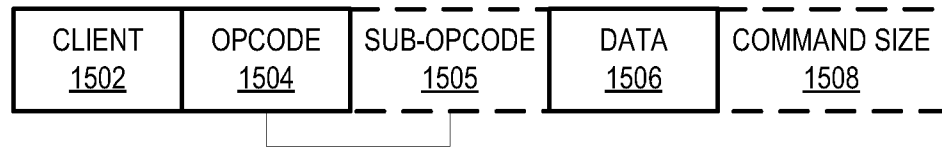
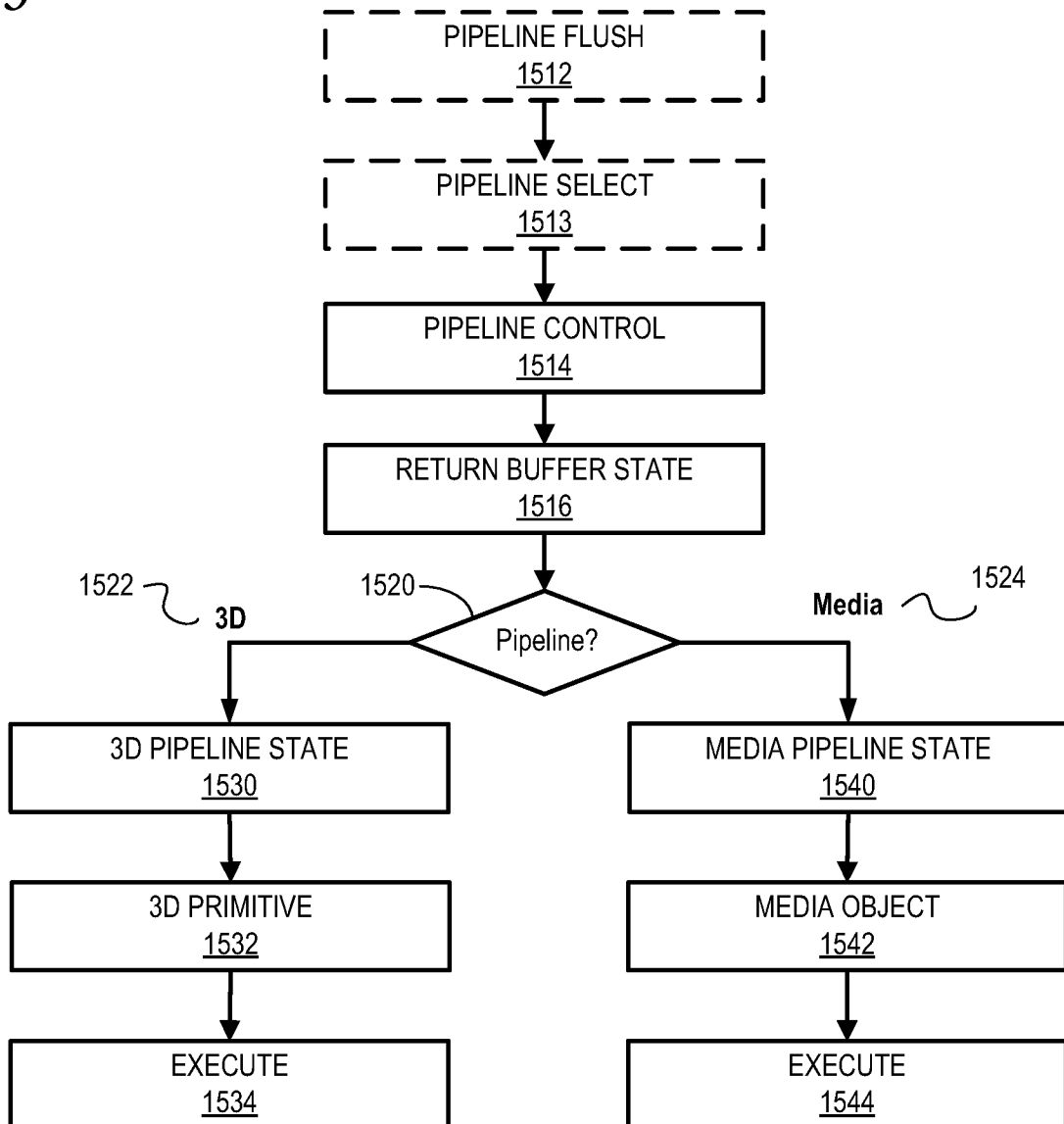
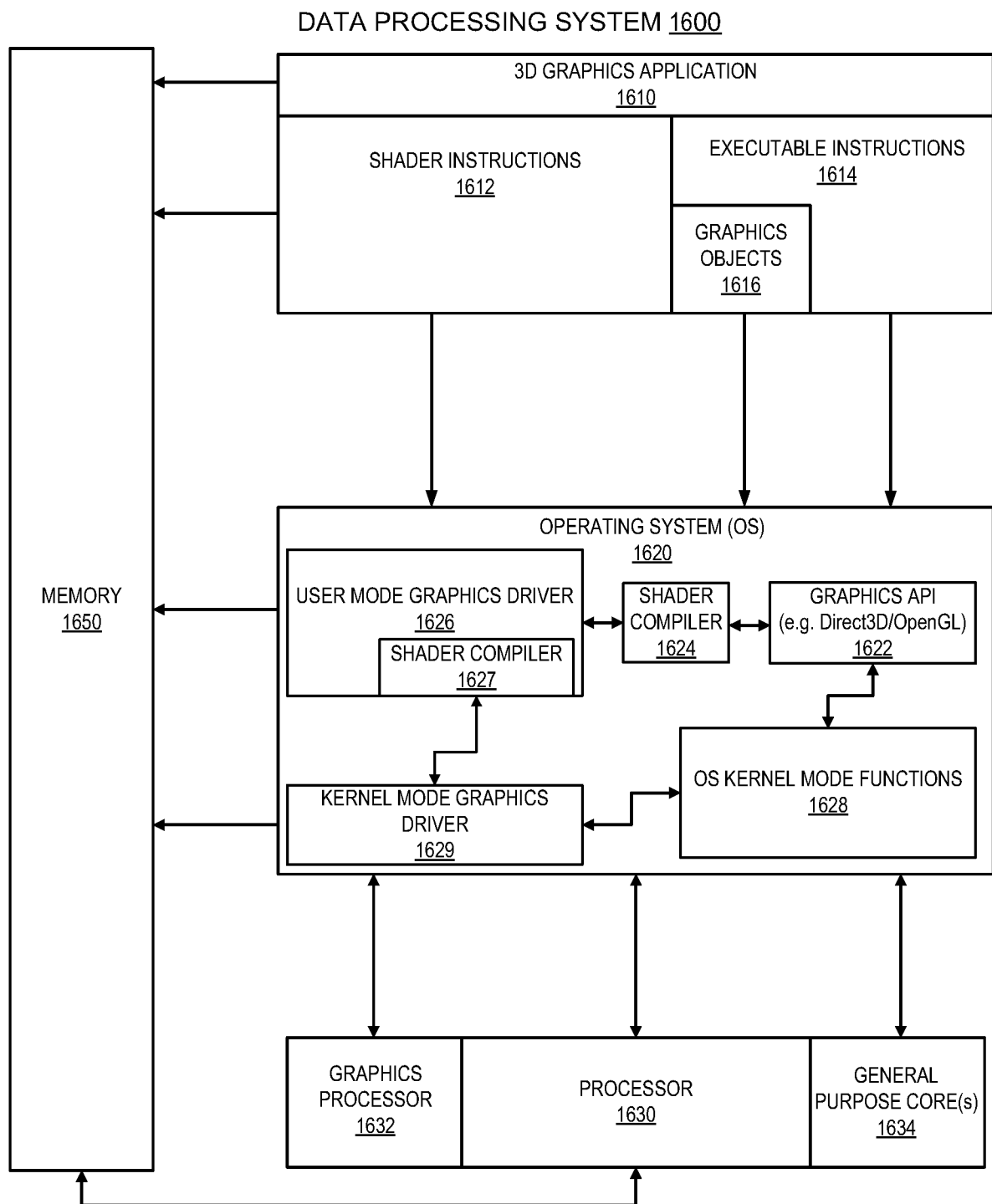


FIG. 15B GRAPHICS PROCESSOR COMMAND SEQUENCE
1510



*FIG. 16*

IP CORE DEVELOPMENT - 1700

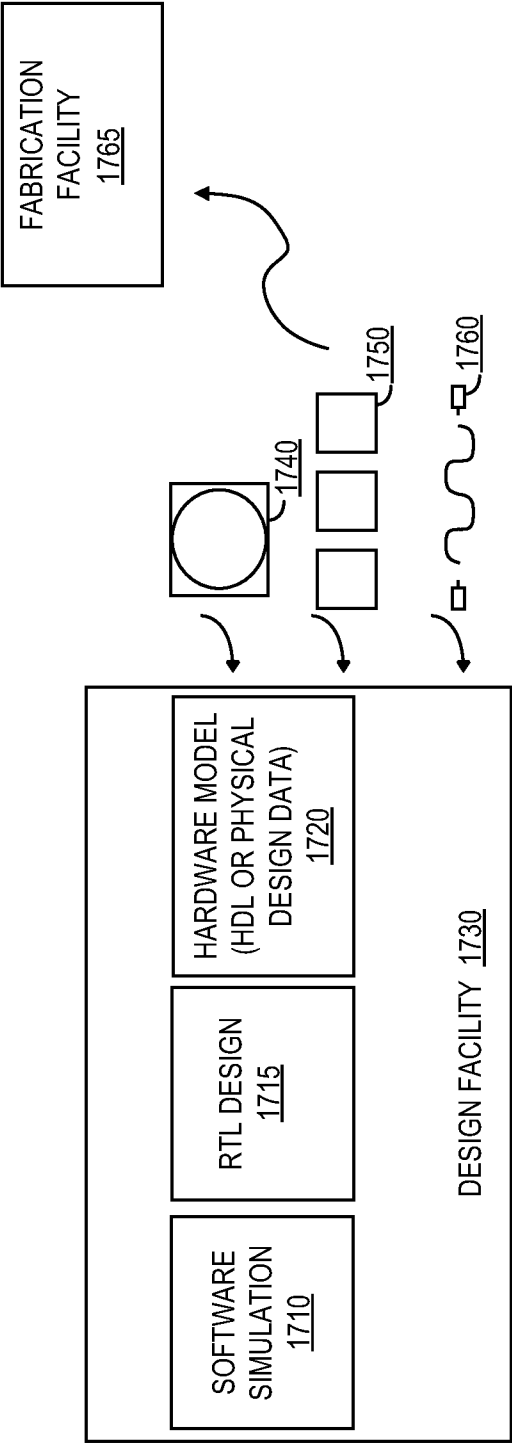
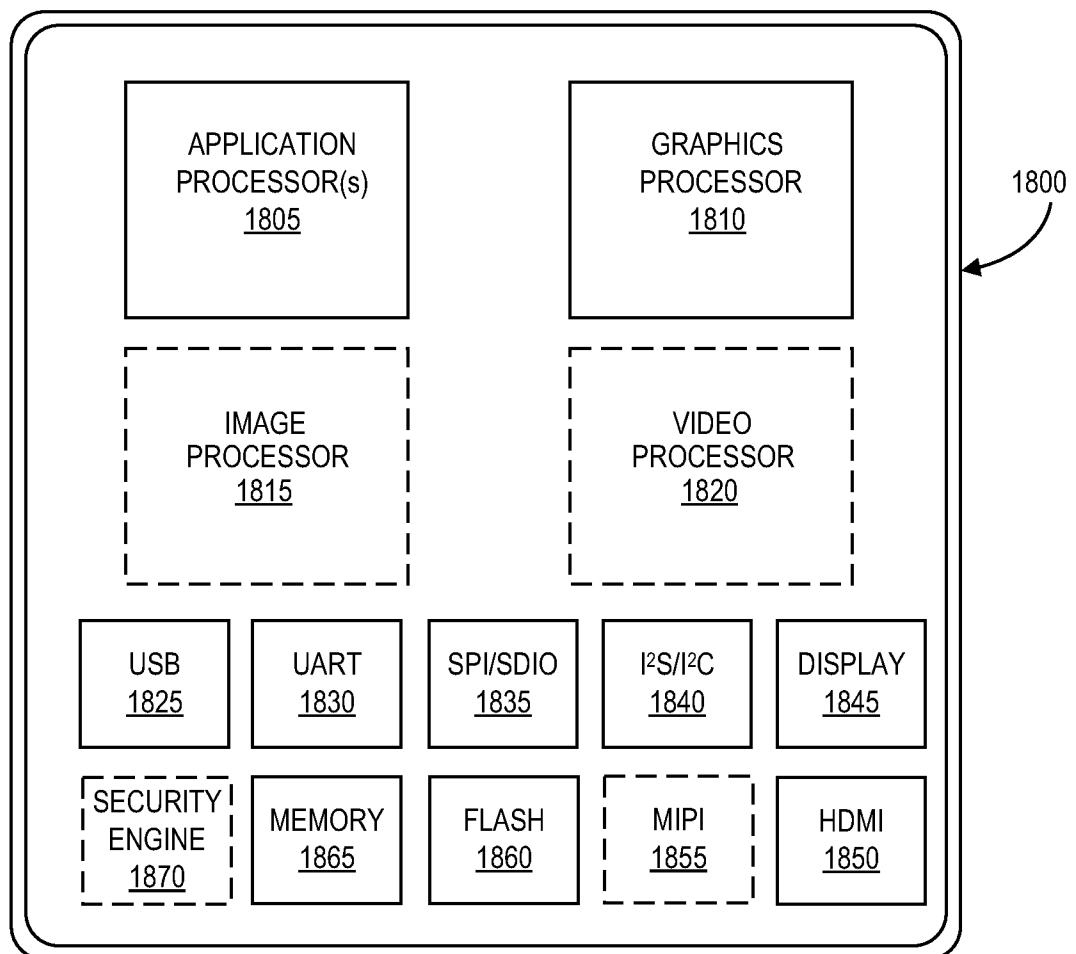


FIG. 17

*Fig. 18*

INTERNATIONAL SEARCH REPORT

International application No.

PCT/CN2015/098934**A. CLASSIFICATION OF SUBJECT MATTER**

G06T 15/50(2006.01)i; G06T 15/10(2006.01)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.; G06F

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

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

CNPAT,CNKI, WPI, EPODOC: surface, render, material, reflect, diffuse, environment, map, illuminate, light, extract, contour, boundary

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2015043806 A1 (ADOBE SYSTEMS INCORPORATED) 12 February 2015 (2015-02-12) description, paragraph [0021] to paragraph [0030], figures 2-3	1-25
A	CN 104574492 A (FUJIAN TIANQING DIGITAL CO., LTD.) 29 April 2015 (2015-04-29) the whole document	1-25
A	US 2011148871 A1 (ELECTRONICS AND TELECOMMUNICATIONS RESEARCH INSTITUTE) 23 June 2011 (2011-06-23) the whole document	1-25
A	US 2005212812 A1 (PIXAR) 29 September 2005 (2005-09-29) the whole document	1-25
A	KR 10-2005-0066501 A (ELECTRONICS&TELECOM RES. INST. ET AL.) 30 June 2005 (2005-06-30) the whole document	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

14 September 2016

Date of mailing of the international search report

22 September 2016

Name and mailing address of the ISA/CN

**STATE INTELLECTUAL PROPERTY OFFICE OF THE
P.R.CHINA**
**6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing
100088, China**

Authorized officer

WU,Huangfei

Facsimile No. (86-10)62019451

Telephone No. (86-10)62413982

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.

PCT/CN2015/098934

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	2015043806	A1	12 February 2015	US	9299188	B2	29 March 2016
				US	2016171755	A1	16 June 2016
CN	104574492	A	29 April 2015	None			
US	2011148871	A1	23 June 2011	KR	20110071767	A	29 June 2011
				KR	101284794	B1	10 July 2013
US	2005212812	A1	29 September 2005	US	7184043	B2	27 February 2007
				WO	2004100072	A1	18 November 2004
				US	7176918	B2	13 February 2007
				US	2005179687	A1	18 August 2005
				US	2005212795	A1	29 September 2005
				US	7443394	B2	28 October 2008
				US	2006250396	A1	09 November 2006
				US	7436404	B2	14 October 2008
				US	2009109221	A1	30 April 2009
				US	7864176	B2	04 January 2011
				US	2009109220	A1	30 April 2009
				US	7859530	B2	28 December 2010
				US	2004217957	A1	04 November 2004
				US	7019744	B2	28 March 2006
				AU2003257117	A1		26 November 2004
KR	10-2005-0066501	A	30 June 2005	KR	100521413	B1	17 October 2005