US 20180061122A1

(19) **United States**

(12) **Patent Application Publication** (10) Pub. No.: US 2018/0061122 A1

CLARBERG (43) **Pub. Date:** **Mar. 1, 2018**

(54) **MULTI-RESOLUTION DEFERRED SHADING USING TEXEL SHADERS IN COMPUTING ENVIRONMENTS**

(71) Applicant: **INTEL CORPORATION**, Santa Clara, CA (US)

(72) Inventor: **FRANZ PETRIK CLARBERG**, Lund (SE)
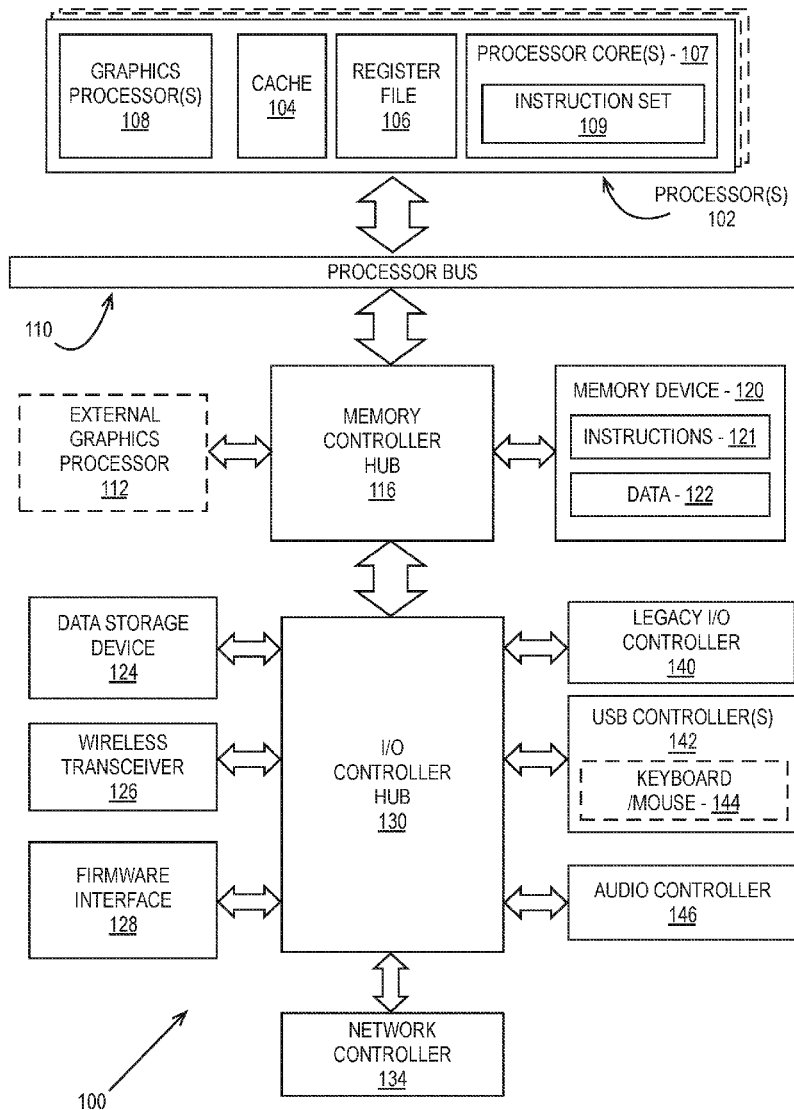
(57) **ABSTRACT**

A mechanism is described for facilitating multi-resolution deferred shading using texel shaders in computing environments. A method of embodiments, as described herein, includes facilitating computation of shading rate in a first pass in a graphics pipeline, where the shading rate relates to a plurality of pixels. The method may further include facilitating texel shading operations in a second pass using the shading rate, where the first pass is performed separate from and prior to the second pass.

GRAPHICS PROCESSOR(S) 108

CACHE 104

REGISTER FILE 106

PROCESSOR CORE(S) - 107

INSTRUCTION SET 109

PROCESSOR(S) 102

PROCESSOR BUS

110

EXTERNAL GRAPHICS PROCESSOR 112

MEMORY CONTROLLER HUB 116

MEMORY DEVICE - 120

INSTRUCTIONS - 121

DATA - 122

DATA STORAGE DEVICE 124

WIRELESS TRANSCEIVER 126

FIRMWARE INTERFACE 128

I/O CONTROLLER HUB 130

LEGACY I/O CONTROLLER 140

USB CONTROLLER(S) 142

KEYBOARD /MOUSE - 144

AUDIO CONTROLLER 146

NETWORK CONTROLLER 134

100

FIG. 1

PROCESSOR 200

BUS CONTROLLER UNIT(S) 216

DISPLAY CONTROLLER 211

SYSTEM AGENT CORE 210

MEMORY CONTROLLER 214

CORE 202N

CACHE UNIT(S) 204N

SHARED CACHE UNIT(S) - 206

RING - 212

CORE 202A

CACHE UNIT(S) 204A

GRAPHICS PROCESSOR 208

I/O 213

EMBEDDED MEMORY MODULE 218

FIG. 2

GRAPHICS PROCESSOR 300

GRAPHICS PROCESSING ENGINE 310

DISPLAY CONTROLLER 302

BLIT ENGINE 304

3D PIPELINE 312

3D/MEDIA SUB-SYSTEM 315

MEDIA PIPELINE 316

VIDEO CODEC ENGINE 306

MEMORY INTERFACE - 314

DISPLAY DEVICE 320

FIG. 3

GRAPHICS PROCESSING ENGINE
410

421 SAMPLER

422 MATH

423 INTER-THREAD COMMUNICATION

...

425 CACHE(S)

SHARED FUNCTION LOGIC
420

GRAPHICS CORE ARRAY
414

UNIFIED RETURN BUFFER
418

3D PIPELINE
312

MEDIA PIPELINE
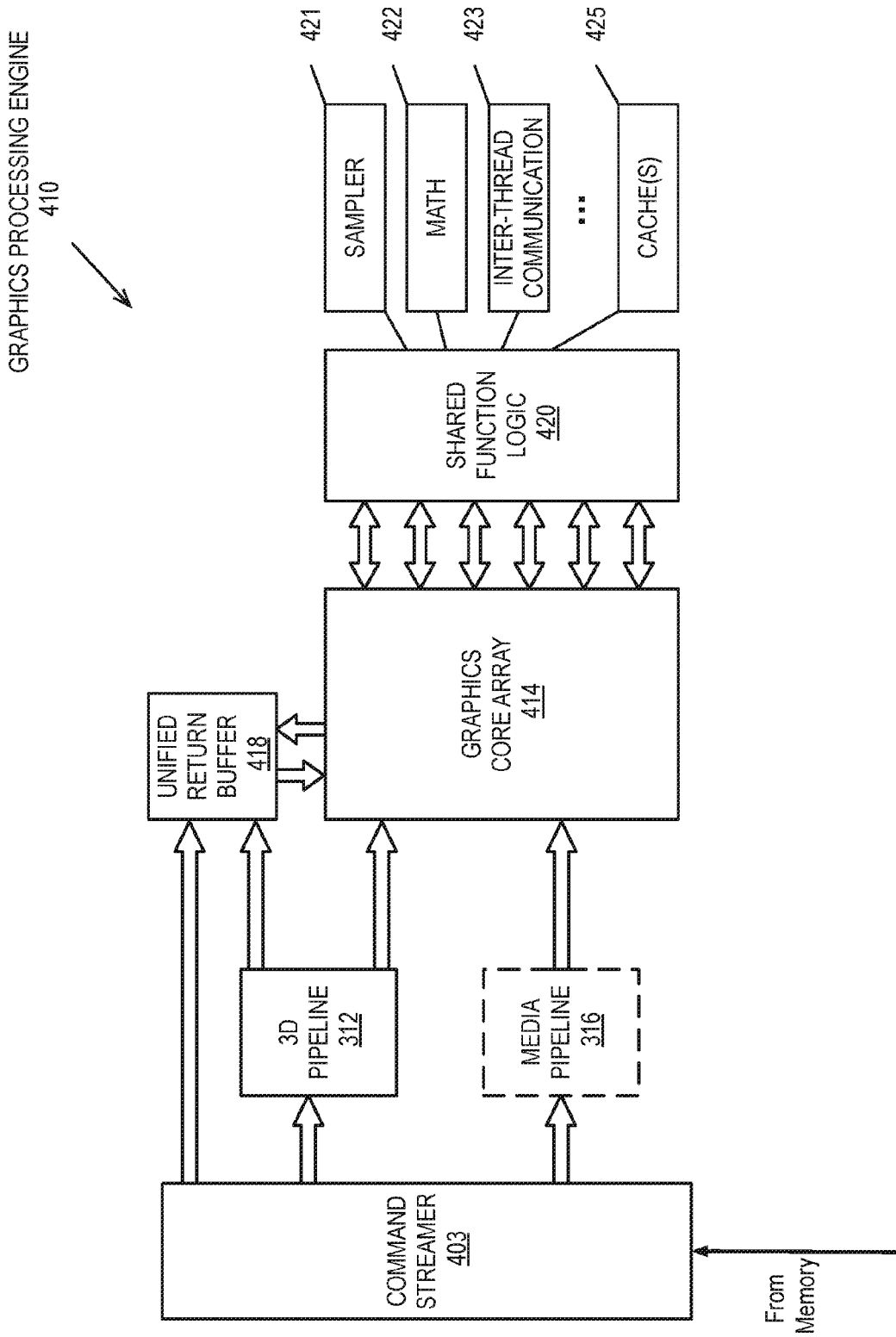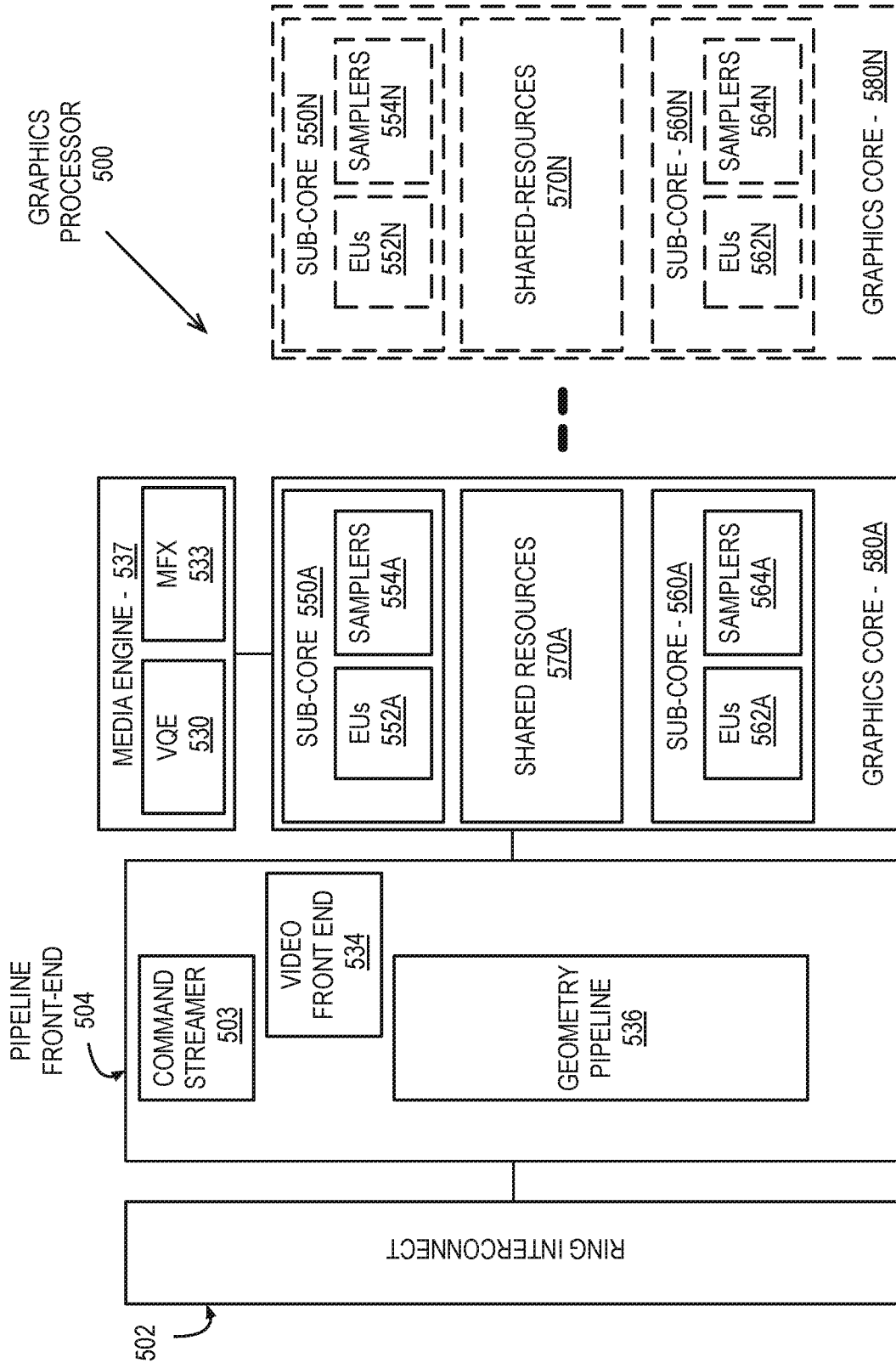316

COMMAND STREAMER
403

From Memory

FIG. 4

FIG. 5

FIG. 6

GRAPHICS PROCESSOR INSTRUCTION FORMATS
700

128-BIT INSTRUCTION
710

OPCODE 712	CONTROL 714	EXEC-SIZE 716	DEST 718	SRC0 720	SRC1 722	SRC2 724	ACCESS/ADDRESS MODE 726

64-BIT COMPACT INSTRUCTION
730

OPCODE 712	INDEX 713	CONTROL 714	DEST 718	SRC0 720	SRC1 722

OPCODE DECODE
740

7	6	5	4	3	2	1	0

opcode=000xxxxxb → Move/Logic - 742

opcode=0010xxxxb → Flow Control - 744

opcode=0011xxxxb → Miscellaneous - 746

opcode=0100xxxxb → Parallel Math - 748

opcode=0101xxxxb → Vector Math - 750

FIG. 7

GRAPHICS PROCESSOR
800

MEDIA PIPELINE
830

DISPLAY ENGINE
840

COMMAND
STREAMER
803

GRAPHICS
PIPELINE
820

802

VIDEO
FRONT-END
834

MEDIA
ENGINE
837

2D ENGINE
841

DISPLAY
CONTROLLER
843

VERTEX
FETCHER
805

831

EXECUTION LOGIC
850

VERTEX
SHADER
807

RING INTERCONNECT

THREAD DISPATCHER

L1
CACHE
851

EXECUTION
UNITS
852A

EXECUTION
UNITS
852B

SAMPLER
854

DATA
PORT
856

TEXTURE
CACHE
858

HULL
SHADER
811

TESSELLATOR
813

DOMAIN
SHADER
817

GEOMETRY
SHADER
819

RASTER/
DEPTH
873

L3
CACHE
875

PIXEL
OPS
877

RENDER
CACHE
878

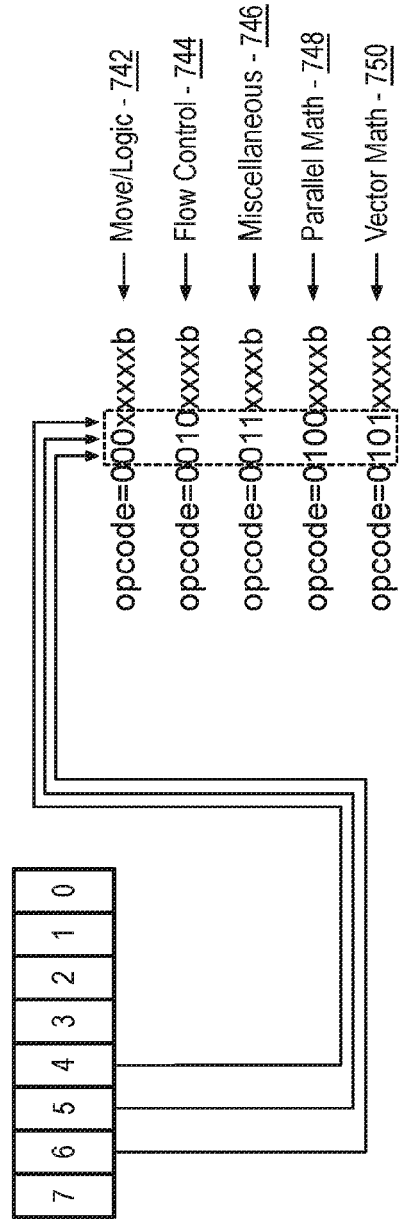DEPTH
CACHE
879

STREAM
OUT
823

RENDER OUTPUT
PIPELINE
870

CLIP/
SETUP
829

FIG. 8

FIG. 9A   GRAPHICS PROCESSOR COMMAND FORMAT
900

CLIENT 902	OPCODE 904	SUB-OPCODE 905	DATA 906	COMMAND SIZE 908

FIG. 9B   GRAPHICS PROCESSOR COMMAND SEQUENCE
910

PIPELINE FLUSH
912

PIPELINE SELECT
913

PIPELINE CONTROL
914

RETURN BUFFER STATE
916

922   920   Media   924

3D   Pipeline?

| 3D PIPELINE STATE 930 | MEDIA PIPELINE STATE 940 |

| 3D PRIMITIVE 932 | MEDIA OBJECT 942 |

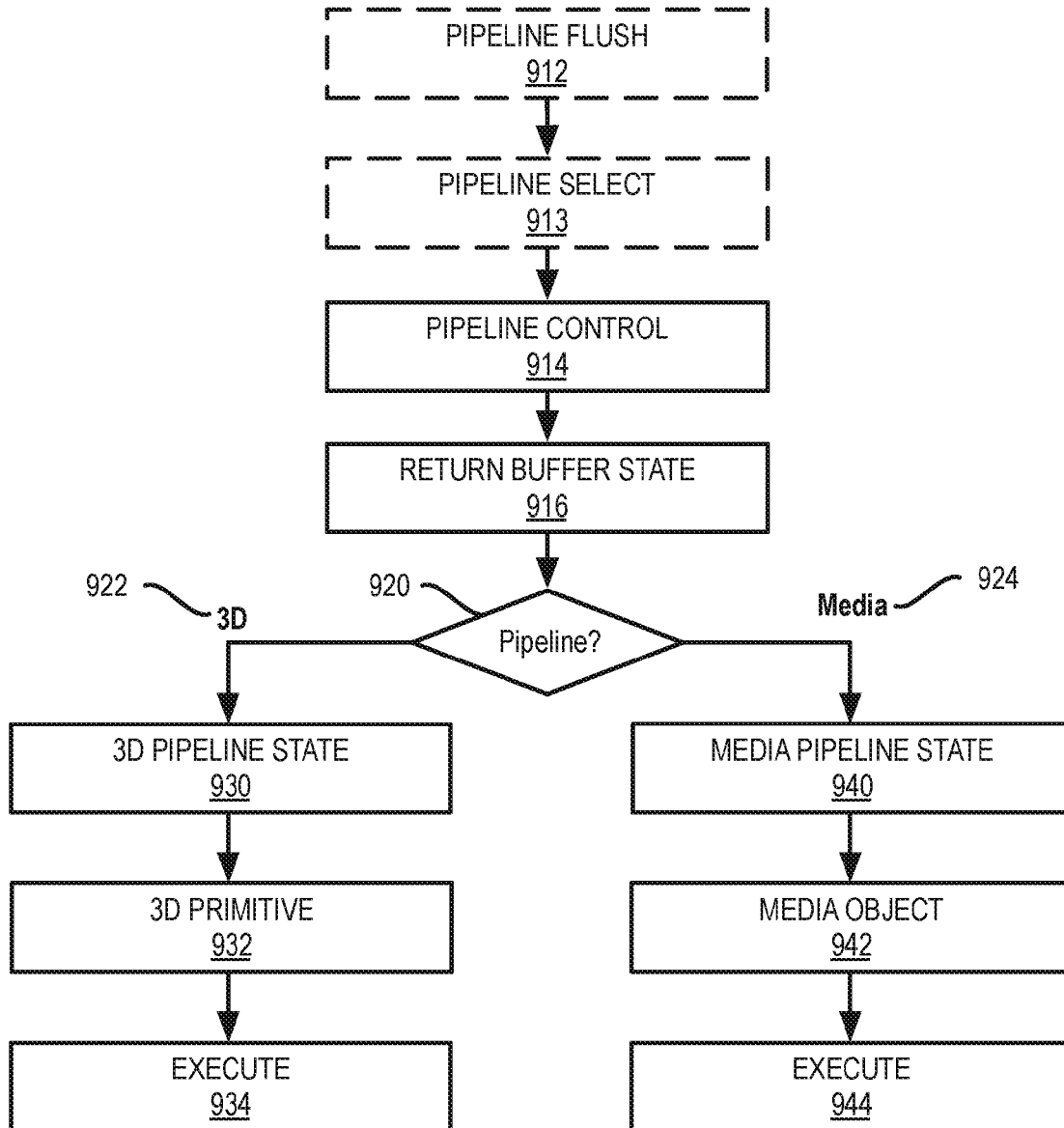| EXECUTE 934 | EXECUTE 944 |

DATA PROCESSING SYSTEM -1000

FIG. 10

IP CORE DEVELOPMENT - 1100

FABRICATION FACILITY 1165

NON-VIOLATILE MEMORY 1140

WIRED CONNECTION 1150

WIRELESS CONNECTION 1160

SOFTWARE SIMULATION 1110

SIMULATION MODEL 1112

REGISTER TRANSFER LEVEL DESIGN 1115

HARDWARE MODEL (HDL OR PHYSICAL DESIGN DATA) 1120

DESIGN FACILITY 1130

FIG. 11

SOC
INTEGRATED CIRCUIT
1200

APPLICATION
PROCESSOR(s)
1205

GRAPHICS
PROCESSOR
1210

IMAGE
PROCESSOR
1215

VIDEO
PROCESSOR
1220

USB
1225

UART
1230

SPI/SDIO
1235

I^2S/I^2C
1240

DISPLAY
1245

SECURITY
ENGINE
1270

MEMORY
1265

FLASH
1260

MIPI
1255

HDMI
1250

FIG. 12

GRAPHICS PROCESSOR
1310

VERTEX PROCESSOR
1305

| FRAGMENT PROCESSOR 1315A | FRAGMENT PROCESSOR 1315C | ▬ ▬ | FRAGMENT PROCESSOR 1315N-1 |

| FRAGMENT PROCESSOR 1315B | FRAGMENT PROCESSOR 1315D | ▬ ▬ | FRAGMENT PROCESSOR 1315N |

| MMU 1320A | MMU 1320B |

| CACHE 1325A | CACHE 1325B |

| INTERCONNECT 1330A | INTERCONNECT 1330B |

FIG. 13

GRAPHICS PROCESSOR
1410

INTER-CORE TASK-MANAGER
(e.g., THREAD DISPATCHER)
1405

| SHADER CORE 1415A | SHADER CORE 1415C | SHADER CORE 1415E | - - | SHADER CORE 1415N-1 |

| SHADER CORE 1415B | SHADER CORE 1415D | SHADER CORE 1415F | - - | SHADER CORE 1415N |

TILING UNIT
1418

| MMU 1320A | MMU 1320B |

| CACHE 1325A | CACHE 1325B |

| INTERCONNECT 1330A | INTERCONNECT 1330B |

FIG. 14

FIG. 15

COMPUTING DEVICE (e.g., HOST MACHINE)
1500

OPERATING SYSTEM (OS)
1506

GRAPHICS DRIVER
1516

MULTI-RESOLUTION DEFERRED SHADING
MECHANISM
1510

GRAPHICS PROCESSING
UNIT (GPU)
1514

CENTRAL PROCESSING
UNIT (CPU)

1512

MEMORY

1508

INPUT/OUTPUT (I/O) SOURCE(S)
(e.g., CAMERA(S), MICROPROCESSOR(S),
SPEAKER(S), SENSOR(S), DISPLAY SCREEN(S),
MEDIA PLAYER(S), ETC.)

1504

FIG. 16

COMPUTING DEVICE (e.g., COMMUNICATON DEVICES, DATA PROCESSING DEVICES, MOBILE COMPUTERS, SMART DEVICES, GAMING DEVICES, ETC.)
1500

GRAPHICS DRIVER
1516

MULTI-RESOLUTION DEFERRED SHADING MECHANISM
1510

DETECTION/RECEPTION LOGIC
1601

DIVISION/SPLITTING LOGIC
1605

PIXEL SHADING COMPUTATION LOGIC
1607

SHADING PASS LOGIC
1609

EXECUTION/FORWARDING LOGIC
1611

COMMUNICATION/COMPATIBILTY LOGIC
1613

CENTRAL PROCESSING UNIT (CPU)
1512

GRAPHICS PROCESSING UNIT (GPU)
1514

1625

COMMUNICATION MEDIUM(S) (e.g., CLOUD NETWORK, INTERNET, PROXIMITY NETWORK, ETC.)
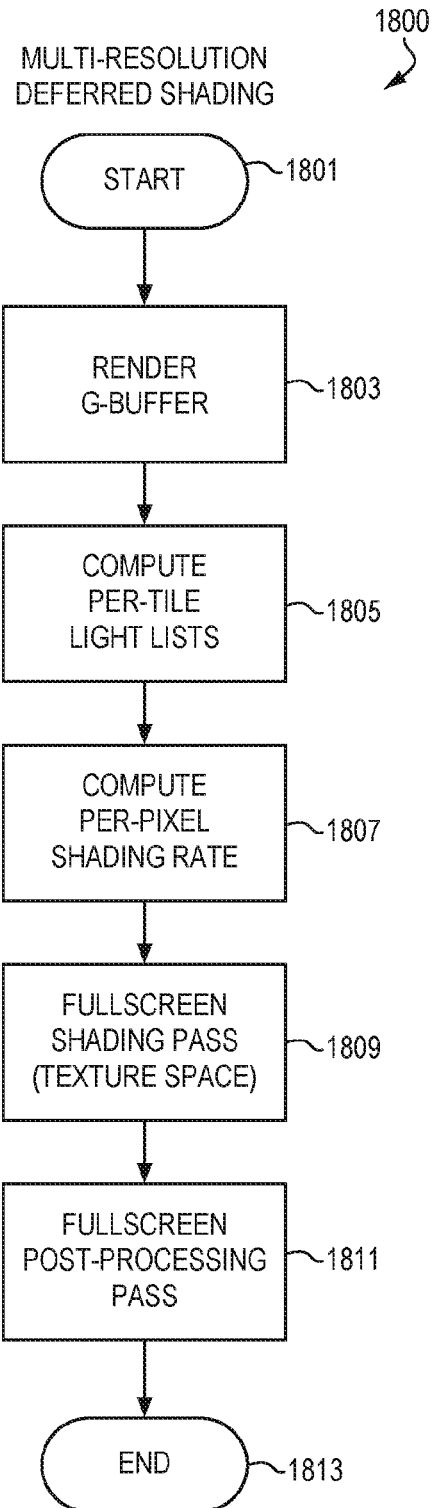
DATABASE(S)
1630

FIG. 17A
PRIOR ART

FIG. 17B
PRIOR ART

TRADITIONAL
DEFERRRED SHADING

1730

START ~1731

RENDER
G-BUFFER ~1733

COMPUTE
PER-TILE
LIGHT LISTS ~1735

FULLSCREEN
SHADING PASS
(SCREEN SPACE) ~1737

FULLSCREEN
POST-PROCESSING
PASS ~1739

END ~1741

FIG. 18A

MULTI-RESOLUTION
DEFERRED SHADING

1800

START ~1801

RENDER
G-BUFFER ~1803

COMPUTE
PER-TILE
LIGHT LISTS ~1805

COMPUTE
PER-PIXEL
SHADING RATE ~1807

FULLSCREEN
SHADING PASS
(TEXTURE SPACE) ~1809

FULLSCREEN
POST-PROCESSING
PASS ~1811

END ~1813

TRADITIONAL DEFERRED SHADING

1750

START — 1751
1753 — DRAW OPAQUE
1763 — G-BUFFER
1755 — PER-PIXEL SHADING G-BUFFER
1757 — DRAW ALPHA
1765 — LIGHT LISTS
1759 — POST PROCESS
1761 — DISPLAY

FIG. 17C
PRIOR ART

MULTI-RESOLUTION DEFERRED SHADING USING THE TEXEL SHADER

1821 — START
1823 — DRAW OPAQUE
1839 — G-BUFFER
1825 — COMPUTE PER-PIXEL SHADING RATE

FULLSCREEN SHADING PASS (TEXTURE SPACE) — 1845

1827 — TRIGGER SHADING
EVALUATE
1831 — PER-TEXEL SHADING G-BUFFER
1829 — FETCH SHADING
SAMPLE
1841 — PT
1833 — DRAW ALPHA
1843 — LIGHT LISTS
1835 — POST PROCESS
1837 — DISPLAY

FIG. 18B

FIG. 18C

1850

COMPUT PER-PIXEL SHADING RATE

START ~1851

LOAD EXTERNAL
INPUT
(EYE TRACKING DATA, . .) ~1853

FOR EACH PIXEL (x,y)
ON SCREEN ~1855

LOAD ATTRIBUTES AT (x,y)
FROM G-BUFFER ~1857

COMPUTE SHADING RATE
(lod) BASED ON
ATTRIBUTES AND/OR
EXTERNAL INPUT ~1859

STORE SHADING RATE
lod AT PIXEL (x,y) ~1861

MORE
PIXELS? ~1863

YES

NO

DONE ~1865

1870

FULLSCREEN SHADING PASS (TEXTURE SPACE)

1871
a) TRIGGER TEXEL SHADING

START ~1873

b) FETCH TEXEL SHADING ~1893

FOR EACH PIXEL (x,y) ON SCREEN ~1875

LOAD SHADING RATE (LOD) ~1877

COMPUTE FILTER FOOTPRINT BASED ON (x, y, lod) ~1879

FOR EACH TEXEL (u, v) IN FILTER FOOTPRINT ~1881

IS TEXEL (u, v) SHADED? ~1883 — YES

NO

SHADE TEXEL (USING G-BUFFER ATTRIBUTES, LIGHT LISTS, ..) ~1885

STORE SHADING TO PROCEDURAL TEXTURE (PT) ~1887

MORE TEXELS? ~1889 YES

NO

MORE PIXELS? ~1891 YES   NO

FOR EACH PIXEL (x,y) ON SCREEN ~1894

LOAD SHADING RATE (LOD) ~1895

SAMPLE TEXEL SHADING AT (x, y, lod) TO OBTAIN FILTERED COLOR ~1896

STORE COLOR TO PIXEL (x, y) IN FRAME BUFFER ~1897

MORE PIXELS? ~1898   YES

NO

DONE ~1899

FIG. 18D

MULTI-RESOLUTION DEFERRED SHADING USING TEXEL SHADERS IN COMPUTING ENVIRONMENTS

FIELD

[0001] Embodiments described herein generally relate to computers. More particularly, embodiments are described for facilitating multi-resolution deferred shading using texel shaders in computing environments.

BACKGROUND

[0002] Three-dimensional (3D) computing games and benchmarks typically spend a majority of the frame time on computing the appearance (shading) of each pixel, where shading is determined based on material properties and light sources. These lighting computations are often expensive, such as in order of hundreds or even thousands of shader instructions per pixel. With high-resolution displays, next-generation virtual reality (VR) headsets, etc., having both high resolution and high refresh rates, the shading cost becomes prohibitive for low/medium-powered graphics devices. Further, with eye tracking hardware becoming more widespread, such as in case of next-generation VR/augmented reality (AR) computing devices, their shading rate continues to be significantly high even in the periphery where the user is not looking. In order to reduce the shading cost and make rendering on such devices feasible, it is desirable to exploit certain characteristics of the rendered image to avoid/reduce expensive computations, such as often large parts of a rendered image are smooth or of low contrast.

[0003] Deferred shading (also known as deferred lighting) is a prevalent rendering technique is today's application; however, with this technique, applications uniformly lower the rendering/shading resolution and then up-scale the relevant images before display. This results in substantial reduction in image quality as image features, including sharp edges and high-frequency details, are under-sampled.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] Embodiments are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

[0005] FIG. 1 is a block diagram of a processing system, according to an embodiment.

[0006] FIG. 2 is a block diagram of an embodiment of a processor having one or more processor cores, an integrated memory controller, and an integrated graphics processor.

[0007] FIG. 3 is a block diagram of a graphics processor, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores.

[0008] FIG. 4 is a block diagram of a graphics processing engine of a graphics processor in accordance with some embodiments.

[0009] FIG. 5 is a block diagram of another embodiment of a graphics processor.

[0010] FIG. 6 illustrates thread execution logic including an array of processing elements employed in some embodiments of a graphics processing engine.

[0011] FIG. 7 is a block diagram illustrating a graphics processor instruction formats according to some embodiments.

[0012] FIG. 8 is a block diagram of another embodiment of a graphics processor.

[0013] FIG. 9A is a block diagram illustrating a graphics processor command format according to an embodiment.

[0014] FIG. 9B is a block diagram illustrating a graphics processor command sequence according to an embodiment.

[0015] FIG. 10 illustrates exemplary graphics software architecture for a data processing system according to some embodiments.

[0016] FIG. 11 is a block diagram illustrating an IP core development system that may be used to manufacture an integrated circuit to perform operations according to an embodiment.

[0017] FIG. 12 is a block diagram illustrating an exemplary system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

[0018] FIG. 13 is a block diagram illustrating an exemplary graphics processor of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

[0019] FIG. 14 is a block diagram illustrating an additional exemplary graphics processor of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

[0020] FIG. 15 illustrates a computing device employing a multi-resolution deferred shading mechanism according to one embodiment.

[0021] FIG. 16 illustrates a multi-resolution deferred shading mechanism according to one embodiment.

[0022] FIG. 17A illustrates a conventional graphics pipeline implementing texel shading.

[0023] FIG. 17B illustrates a conventional method for deferred shading.

[0024] FIG. 17C illustrates a conventional transaction sequence for deferred shading.

[0025] FIG. 18A illustrates a method for multi-resolution deferred shading according to one embodiment.

[0026] FIG. 18B illustrates a transaction sequence for multi-resolution deferred shading using a texel shader according to one embodiment.

[0027] FIG. 18C illustrates a method for computing per-pixel shading rate according to one embodiment.

[0028] FIG. 18D illustrates a method for performing fullscreen shading pass (texture space) according to one embodiment.

DETAILED DESCRIPTION

[0029] In the following description, numerous specific details are set forth. However, embodiments, as described herein, may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in details in order not to obscure the understanding of this description.

[0030] Embodiment provide for a novel technique of multi-resolution deferred shading in a single pass by selectively reducing shading resolution to continuously maintain high quality of imaging results. This novel technique leverages the texel shader stage in the graphics pipeline to allow for greater accuracy and efficiency in rendering of images at high-resolution displays, VR/AR devices, etc. Further, this

novel technique may facilitate use and exploitation of hardware/application programming interface (API)-based support for texel shaders in a different manner than it is typically intended or used.

[0031] This novel multi-resolution shading technique reduces the shading costs and makes rendering feasible on certain devices (such as next-generation VR/AR computing devices) by selectively reducing the resolution at which shading computations are done.

[0032] For example, instead of performing deferred shading calculations in a shader running once per pixel, in one embodiment, calculations are performed in a shader running once per texel (e.g., texel shader) on a fullscreen texture. By varying the mipmap level (e.g., Level of Detail ("LOD" or "lod")), the shading rate is effectively controlled and the cost of shading each texel is amortized over multiple pixels.

[0033] In graphics computing, a procedural texture (PT) refers to a computer-generated image created using an algorithm to render a realistic representation of a material, where procedural texture may be regarded as a generalization of traditional memory-based textures to allow a texel shader (TS) to generate texels. Texel shading, using a texel shader, is in some embodiments regarded as a 1-pass model where the texel shader runs synchronously, such as issuing pixel shader thread waits for the texel shader thread(s) to finish and return a shaded result. In other embodiments, texel shading is regarded as a 2-pass model, where the texel shader runs asynchronously. In that case, the pixel shader thread issues a request to shade a group of texels, but it does not wait for the texel shader thread(s) to finish.

[0034] It is contemplated that terms like "request", "query", "job", "work", "work item", and "workload" may be referenced interchangeably throughout this document. Similarly, an "application" or "agent" may refer to or include a computer program, a software application, a game, a workstation application, etc., offered through an application programming interface (API), such as a free rendering API, such as Open Graphics Library (OpenGL®), DirectX® 11, DirectX® 12, etc., where "dispatch" may be interchangeably referred to as "work unit" or "draw" and similarly, "application" may be interchangeably referred to as "workflow" or simply "agent". For example, a workload, such as that of a three-dimensional (3D) game, may include and issue any number and type of "frames" where each frame may represent an image (e.g., sailboat, human face). Further, each frame may include and offer any number and type of work units, where each work unit may represent a part (e.g., mast of sailboat, forehead of human face) of the image (e.g., sailboat, human face) represented by its corresponding frame. However, for the sake of consistency, each item may be referenced by a single term (e.g., "dispatch", "agent", etc.) throughout this document.

[0035] In some embodiments, terms like "display screen" and "display surface" may be used interchangeably referring to the visible portion of a display device while the rest of the display device may be embedded into a computing device, such as a smartphone, a wearable device, etc. It is contemplated and to be noted that embodiments are not limited to any particular computing device, software application, hardware component, display device, display screen or surface, protocol, standard, etc. For example, embodiments may be applied to and used with any number and type of real-time applications on any number and type of computers, such as desktops, laptops, tablet computers, smartphones, head-

mounted displays and other wearable devices, and/or the like. Further, for example, rendering scenarios for efficient performance using this novel technique may range from simple scenarios, such as desktop compositing, to complex scenarios, such as 3D games, augmented reality applications, etc.

[0036] System Overview

[0037] FIG. 1 is a block diagram of a processing system 100, according to an embodiment. In various embodiments the system 100 includes one or more processors 102 and one or more graphics processors 108, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 102 or processor cores 107. In one embodiment, the system 100 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

[0038] An embodiment of system 100 can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming console, a handheld game console, or an online game console. In some embodiments system 100 is a mobile phone, smart phone, tablet computing device or mobile Internet device. Data processing system 100 can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, smart eyewear device, augmented reality device, or virtual reality device. In some embodiments, data processing system 100 is a television or set top box device having one or more processors 102 and a graphical interface generated by one or more graphics processors 108.

[0039] In some embodiments, the one or more processors 102 each include one or more processor cores 107 to process instructions which, when executed, perform operations for system and user software. In some embodiments, each of the one or more processor cores 107 is configured to process a specific instruction set 109. In some embodiments, instruction set 109 may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). Multiple processor cores 107 may each process a different instruction set 109, which may include instructions to facilitate the emulation of other instruction sets. Processor core 107 may also include other processing devices, such a Digital Signal Processor (DSP).

[0040] In some embodiments, the processor 102 includes cache memory 104. Depending on the architecture, the processor 102 can have a single internal cache or multiple levels of internal cache. In some embodiments, the cache memory is shared among various components of the processor 102. In some embodiments, the processor 102 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among processor cores 107 using known cache coherency techniques. A register file 106 is additionally included in processor 102 which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). Some registers may be general-purpose registers, while other registers may be specific to the design of the processor 102.

[0041] In some embodiments, processor 102 is coupled with a processor bus 110 to transmit communication signals such as address, data, or control signals between processor

102 and other components in system 100. In one embodiment the system 100 uses an exemplary 'hub' system architecture, including a memory controller hub 116 and an Input Output (I/O) controller hub 130. A memory controller hub 116 facilitates communication between a memory device and other components of system 100, while an I/O Controller Hub (ICH) 130 provides connections to I/O devices via a local I/O bus. In one embodiment, the logic of the memory controller hub 116 is integrated within the processor.

[0042] Memory device 120 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 120 can operate as system memory for the system 100, to store data 122 and instructions 121 for use when the one or more processors 102 executes an application or process. Memory controller hub 116 also couples with an optional external graphics processor 112, which may communicate with the one or more graphics processors 108 in processors 102 to perform graphics and media operations.

[0043] In some embodiments, ICH 130 enables peripherals to connect to memory device 120 and processor 102 via a high-speed I/O bus. The I/O peripherals include, but are not limited to, an audio controller 146, a firmware interface 128, a wireless transceiver 126 (e.g., Wi-Fi, Bluetooth), a data storage device 124 (e.g., hard disk drive, flash memory, etc.), and a legacy I/O controller 140 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to the system. One or more Universal Serial Bus (USB) controllers 142 connect input devices, such as keyboard and mouse 144 combinations. A network controller 134 may also couple with ICH 130. In some embodiments, a high-performance network controller (not shown) couples with processor bus 110. It will be appreciated that the system 100 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 130 may be integrated within the one or more processor 102, or the memory controller hub 116 and I/O controller hub 130 may be integrated into a discreet external graphics processor, such as the external graphics processor 112.

[0044] FIG. 2 is a block diagram of an embodiment of a processor 200 having one or more processor cores 202A-202N, an integrated memory controller 214, and an integrated graphics processor 208. Those elements of FIG. 2 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 200 can include additional cores up to and including additional core 202N represented by the dashed lined boxes. Each of processor cores 202A-202N includes one or more internal cache units 204A-204N. In some embodiments each processor core also has access to one or more shared cached units 206.

[0045] The internal cache units 204A-204N and shared cache units 206 represent a cache memory hierarchy within the processor 200. The cache memory hierarchy may include at least one level of instruction and data cache within each 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 206 and 204A-204N.

[0046] In some embodiments, processor 200 may also include a set of one or more bus controller units 216 and a system agent core 210. The one or more bus controller units 216 manage a set of peripheral buses, such as one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express). System agent core 210 provides management functionality for the various processor components. In some embodiments, system agent core 210 includes one or more integrated memory controllers 214 to manage access to various external memory devices (not shown).

[0047] In some embodiments, one or more of the processor cores 202A-202N include support for simultaneous multi-threading. In such embodiment, the system agent core 210 includes components for coordinating and operating cores 202A-202N during multi-threaded processing. System agent core 210 may additionally include a power control unit (PCU), which includes logic and components to regulate the power state of processor cores 202A-202N and graphics processor 208.

[0048] In some embodiments, processor 200 additionally includes graphics processor 208 to execute graphics processing operations. In some embodiments, the graphics processor 208 couples with the set of shared cache units 206, and the system agent core 210, including the one or more integrated memory controllers 214. In some embodiments, a display controller 211 is coupled with the graphics processor 208 to drive graphics processor output to one or more coupled displays. In some embodiments, display controller 211 may be a separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 208 or system agent core 210.

[0049] In some embodiments, a ring based interconnect unit 212 is used to couple the internal components of the processor 200. However, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques, including techniques well known in the art. In some embodiments, graphics processor 208 couples with the ring interconnect 212 via an I/O link 213.

[0050] The exemplary I/O link 213 represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded memory module 218, such as an eDRAM module. In some embodiments, each of the processor cores 202A-202N and graphics processor 208 use embedded memory modules 218 as a shared Last Level Cache.

[0051] In some embodiments, processor cores 202A-202N are homogenous cores executing the same instruction set architecture. In another embodiment, processor cores 202A-202N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 202A-202N 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 202A-202N 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

200 can be implemented on one or more chips or as an SoC integrated circuit having the illustrated components, in addition to other components.

[0052] FIG. 3 is a block diagram of a graphics processor 300, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. 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 300 includes a memory interface 314 to access memory. Memory interface 314 can be an interface to local memory, one or more internal caches, one or more shared external caches, and/or to system memory.

[0053] In some embodiments, graphics processor 300 also includes a display controller 302 to drive display output data to a display device 320. Display controller 302 includes hardware for one or more overlay planes for the display and composition of multiple layers of video or user interface elements. In some embodiments, graphics processor 300 includes a video codec engine 306 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

[0054] In some embodiments, graphics processor 300 includes a block image transfer (BLIT) engine 304 to perform two-dimensional (2D) rasterizer operations including, for example, bit-boundary block transfers. However, in one embodiment, 2D graphics operations are performed using one or more components of graphics processing engine (GPE) 310. In some embodiments, GPE 310 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

[0055] In some embodiments, GPE 310 includes a 3D pipeline 312 for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). The 3D pipeline 312 includes programmable and fixed function elements that perform various tasks within the element and/or spawn execution threads to a 3D/Media sub-system 315. While 3D pipeline 312 can be used to perform media operations, an embodiment of GPE 310 also includes a media pipeline 316 that is specifically used to perform media operations, such as video post-processing and image enhancement.

[0056] In some embodiments, media pipeline 316 includes fixed function or programmable logic units to perform one or more specialized media operations, such as video decode acceleration, video de-interlacing, and video encode acceleration in place of, or on behalf of video codec engine 306. In some embodiments, media pipeline 316 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 315. The spawned threads perform computations for the media operations on one or more graphics execution units included in 3D/Media sub-system 315.

[0057] In some embodiments, 3D/Media subsystem 315 includes logic for executing threads spawned by 3D pipeline 312 and media pipeline 316. In one embodiment, the pipelines send thread execution requests to 3D/Media subsystem 315, which includes thread dispatch logic for arbitrating and dispatching the various requests to available thread execution resources. The execution resources include an array of graphics execution units to process the 3D and media threads. In some embodiments, 3D/Media subsystem 315 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.

[0058] Graphics Processing Engine

[0059] FIG. 4 is a block diagram of a graphics processing engine 410 of a graphics processor in accordance with some embodiments. In one embodiment, the graphics processing engine (GPE) 410 is a version of the GPE 310 shown in FIG. 3. Elements of FIG. 4 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. For example, the 3D pipeline 312 and media pipeline 316 of FIG. 3 are illustrated. The media pipeline 316 is optional in some embodiments of the GPE 410 and may not be explicitly included within the GPE 410. For example and in at least one embodiment, a separate media and/or image processor is coupled to the GPE 410.

[0060] In some embodiments, GPE 410 couples with or includes a command streamer 403, which provides a command stream to the 3D pipeline 312 and/or media pipelines 316. In some embodiments, command streamer 403 is coupled with memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In some embodiments, command streamer 403 receives commands from the memory and sends the commands to 3D pipeline 312 and/or media pipeline 316. The commands are directives fetched from a ring buffer, which stores commands for the 3D pipeline 312 and media pipeline 316. In one embodiment, the ring buffer can additionally include batch command buffers storing batches of multiple commands. The commands for the 3D pipeline 312 can also include references to data stored in memory, such as but not limited to vertex and geometry data for the 3D pipeline 312 and/or image data and memory objects for the media pipeline 316. The 3D pipeline 312 and media pipeline 316 process the commands and data by performing operations via logic within the respective pipelines or by dispatching one or more execution threads to a graphics core array 414.

[0061] In various embodiments the 3D pipeline 312 can execute one or more shader programs, such as vertex shaders, geometry shaders, pixel shaders, fragment shaders, compute shaders, or other shader programs, by processing the instructions and dispatching execution threads to the graphics core array 414. The graphics core array 414 provides a unified block of execution resources. Multi-purpose execution logic (e.g., execution units) within the graphic core array 414 includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

[0062] In some embodiments the graphics core array 414 also includes execution logic to perform media functions, such as video and/or image processing. In one embodiment, the execution units additionally include general-purpose logic that is programmable to perform parallel general

purpose computational operations, in addition to graphics processing operations. The general purpose logic can perform processing operations in parallel or in conjunction with general purpose logic within the processor core(s) **107** of FIG. **1** or core **202A-202N** as in FIG. **2**.

[0063] Output data generated by threads executing on the graphics core array **414** can output data to memory in a unified return buffer (URB) **418**. The URB **418** can store data for multiple threads. In some embodiments the URB **418** may be used to send data between different threads executing on the graphics core array **414**. In some embodiments the URB **418** may additionally be used for synchronization between threads on the graphics core array and fixed function logic within the shared function logic **420**.

[0064] In some embodiments, graphics core array **414** is scalable, such that the array includes a variable number of graphics cores, each having a variable number of execution units based on the target power and performance level of GPE **410**. In one embodiment the execution resources are dynamically scalable, such that execution resources may be enabled or disabled as needed.

[0065] The graphics core array **414** couples with shared function logic **420** that includes multiple resources that are shared between the graphics cores in the graphics core array. The shared functions within the shared function logic **420** are hardware logic units that provide specialized supplemental functionality to the graphics core array **414**. In various embodiments, shared function logic **420** includes but is not limited to sampler **421**, math **422**, and inter-thread communication (ITC) **423** logic. Additionally, some embodiments implement one or more cache(s) **425** within the shared function logic **420**. A shared function is implemented where the demand for a given specialized function is insufficient for inclusion within the graphics core array **414**. Instead a single instantiation of that specialized function is implemented as a stand-alone entity in the shared function logic **420** and shared among the execution resources within the graphics core array **414**. The precise set of functions that are shared between the graphics core array **414** and included within the graphics core array **414** varies between embodiments.

[0066] FIG. **5** is a block diagram of another embodiment of a graphics processor **500**. Elements of FIG. **5** 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.

[0067] In some embodiments, graphics processor **500** includes a ring interconnect **502**, a pipeline front-end **504**, a media engine **537**, and graphics cores **580A-580N**. In some embodiments, ring interconnect **502** 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.

[0068] In some embodiments, graphics processor **500** receives batches of commands via ring interconnect **502**. The incoming commands are interpreted by a command streamer **503** in the pipeline front-end **504**. In some embodiments, graphics processor **500** includes scalable execution logic to perform 3D geometry processing and media processing via the graphics core(s) **580A-580N**. For 3D geometry processing commands, command streamer **503** supplies commands to geometry pipeline **536**. For at least some

media processing commands, command streamer **503** supplies the commands to a video front end **534**, which couples with a media engine **537**. In some embodiments, media engine **537** includes a Video Quality Engine (VQE) **530** for video and image post-processing and a multi-format encode/decode (MFX) **533** engine to provide hardware-accelerated media data encode and decode. In some embodiments, geometry pipeline **536** and media engine **537** each generate execution threads for the thread execution resources provided by at least one graphics core **580A**.

[0069] In some embodiments, graphics processor **500** includes scalable thread execution resources featuring modular cores **580A-580N** (sometimes referred to as core slices), each having multiple sub-cores **550A-550N**, **560A-560N** (sometimes referred to as core sub-slices). In some embodiments, graphics processor **500** can have any number of graphics cores **580A** through **580N**. In some embodiments, graphics processor **500** includes a graphics core **580A** having at least a first sub-core **550A** and a second sub-core **560A**. In other embodiments, the graphics processor is a low power processor with a single sub-core (e.g., **550A**). In some embodiments, graphics processor **500** includes multiple graphics cores **580A-580N**, each including a set of first sub-cores **550A-550N** and a set of second sub-cores **560A-560N**. Each sub-core in the set of first sub-cores **550A-550N** includes at least a first set of execution units **552A-552N** and media/texture samplers **554A-554N**. Each sub-core in the set of second sub-cores **560A-560N** includes at least a second set of execution units **562A-562N** and samplers **564A-564N**. In some embodiments, each sub-core **550A-550N**, **560A-560N** shares a set of shared resources **570A-570N**. 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.

[0070] Execution Units

[0071] FIG. **6** illustrates thread execution logic **600** including an array of processing elements employed in some embodiments of a GPE. Elements of FIG. **6** 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.

[0072] In some embodiments, thread execution logic **600** includes a shader processor **602**, a thread dispatcher **604**, instruction cache **606**, a scalable execution unit array including a plurality of execution units **608A-608N**, a sampler **610**, a data cache **612**, and a data port **614**. In one embodiment the scalable execution unit array can dynamically scale by enabling or disabling one or more execution units (e.g., any of execution unit **608A**, **608B**, **608C**, **608D**, through **608N-1** and **608N**) based on the computational requirements of a workload. 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 **600** includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache **606**, data port **614**, sampler **610**, and execution units **608A-608N**. In some embodiments, each execution unit (e.g. **608A**) is a stand-alone programmable general purpose computational unit that is capable of executing multiple simultaneous hardware threads while processing multiple data elements in parallel for each thread. In

various embodiments, the array of execution units **608A-608N** is scalable to include any number individual execution units.

[0073] In some embodiments, the execution units **608A-608N** are primarily used to execute shader programs. A shader processor **602** can process the various shader programs and dispatch execution threads associated with the shader programs via a thread dispatcher **604**. In one embodiment the thread dispatcher includes logic to arbitrate thread initiation requests from the graphics and media pipelines and instantiate the requested threads on one or more execution unit in the execution units **608A-608N**. For example, the geometry pipeline (e.g., **536** of FIG. **5**) can dispatch vertex, tessellation, or geometry shaders to the thread execution logic **600** (FIG. **6**) for processing. In some embodiments, thread dispatcher **604** can also process runtime thread spawning requests from the executing shader programs.

[0074] In some embodiments, the execution units **608A-608N** support 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). Each of the execution units **608A-608N** is capable of multi-issue single instruction multiple data (SIMD) execution and multi-threaded operation enables an efficient execution environment in the face of higher latency memory accesses. Each hardware thread within each execution unit has a dedicated high-bandwidth register file and associated independent thread-state. Execution is multi-issue per clock to pipelines capable of integer, single and double precision floating point operations, SIMD branch capability, logical operations, transcendental operations, and other miscellaneous operations. While waiting for data from memory or one of the shared functions, dependency logic within the execution units **608A-608N** causes a waiting thread to sleep until the requested data has been returned. While the waiting thread is sleeping, hardware resources may be devoted to processing other threads. For example, during a delay associated with a vertex shader operation, an execution unit can perform operations for a pixel shader, fragment shader, or another type of shader program, including a different vertex shader.

[0075] Each execution unit in execution units **608A-608N** 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 **608A-608N** support integer and floating-point data types.

[0076] The execution unit instruction set includes 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.

[0077] One or more internal instruction caches (e.g., **606**) are included in the thread execution logic **600** to cache thread instructions for the execution units. In some embodiments, one or more data caches (e.g., **612**) are included to cache thread data during thread execution. In some embodiments, a sampler **610** is included to provide texture sampling for 3D operations and media sampling for media operations. In some embodiments, sampler **610** includes specialized texture or media sampling functionality to process texture or media data during the sampling process before providing the sampled data to an execution unit.

[0078] During execution, the graphics and media pipelines send thread initiation requests to thread execution logic **600** via thread spawning and dispatch logic. Once a group of geometric objects has been processed and rasterized into pixel data, pixel processor logic (e.g., pixel shader logic, fragment shader logic, etc.) within the shader processor **602** is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In some embodiments, a pixel shader or fragment shader calculates the values of the various vertex attributes that are to be interpolated across the rasterized object. In some embodiments, pixel processor logic within the shader processor **602** then executes an application programming interface (API)-supplied pixel or fragment shader program. To execute the shader program, the shader processor **602** dispatches threads to an execution unit (e.g., **608A**) via thread dispatcher **604**. In some embodiments, pixel shader **602** uses texture sampling logic in the sampler **610** to access texture data in texture maps stored in memory. Arithmetic operations on the texture data and the input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

[0079] In some embodiments, the data port **614** provides a memory access mechanism for the thread execution logic **600** output processed data to memory for processing on a graphics processor output pipeline. In some embodiments, the data port **614** includes or couples to one or more cache memories (e.g., data cache **612**) to cache data for memory access via the data port.

[0080] FIG. **7** is a block diagram illustrating a graphics processor instruction formats **700** 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 **700** 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.

[0081] In some embodiments, the graphics processor execution units natively support instructions in a 128-bit instruction format **710**. A 64-bit compacted instruction format **730** is available for some instructions based on the

selected instruction, instruction options, and number of operands. The native 128-bit instruction format **710** provides access to all instruction options, while some options and operations are restricted in the 64-bit format **730**. The native instructions available in the 64-bit format **730** vary by embodiment. In some embodiments, the instruction is compacted in part using a set of index values in an index field **713**. The execution unit hardware references a set of compaction tables based on the index values and uses the compaction table outputs to reconstruct a native instruction in the 128-bit instruction format **710**.

[0082] For each format, instruction opcode **712** defines the operation that the execution unit is to perform. The execution units execute each instruction in parallel across the multiple data elements of each operand. For example, in response to an add instruction the execution unit performs a simultaneous add operation across each color channel representing a texture element or picture element. By default, the execution unit performs each instruction across all data channels of the operands. In some embodiments, instruction control field **714** enables control over certain execution options, such as channels selection (e.g., predication) and data channel order (e.g., swizzle). For instructions in the 128-bit instruction format **710** an exec-size field **716** limits the number of data channels that will be executed in parallel. In some embodiments, exec-size field **716** is not available for use in the 64-bit compact instruction format **730**.

[0083] Some execution unit instructions have up to three operands including two source operands, src0 **720**, src1 **722**, and one destination **718**. 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 **724**), where the instruction opcode **712** 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.

[0084] In some embodiments, the 128-bit instruction format **710** includes an access/address mode field **726** 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.

[0085] In some embodiments, the 128-bit instruction format **710** includes an access/address mode field **726**, which specifies an address mode and/or an access mode for the instruction. In one embodiment the access mode is used 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 may use byte-aligned addressing for source and destination operands and when in a second mode, the instruction may use 16-byte-aligned addressing for all source and destination operands.

[0086] In one embodiment, the address mode portion of the access/address mode field **726** determines whether the instruction is to use direct or indirect addressing. When direct register addressing mode is used bits in the instruction 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.

[0087] In some embodiments instructions are grouped based on opcode **712** bit-fields to simplify Opcode decode **740**. For an 8-bit opcode, bits 4, 5, and 6 allow the execution unit to determine the type of opcode. The precise opcode grouping shown is merely an example. In some embodiments, a move and logic opcode group **742** includes data movement and logic instructions (e.g., move (mov), compare (cmp)). In some embodiments, move and logic group **742** 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 **744** (e.g., call, jump (jmp)) includes instructions in the form of 0010xxxxb (e.g., 0x20). A miscellaneous instruction group **746** includes a mix of instructions, including synchronization instructions (e.g., wait, send) in the form of 0011xxxxb (e.g., 0x30). A parallel math instruction group **748** includes component-wise arithmetic instructions (e.g., add, multiply (mul)) in the form of 0100xxxxb (e.g., 0x40). The parallel math group **748** performs the arithmetic operations in parallel across data channels. The vector math group **750** includes arithmetic instructions (e.g., dp4) in the form of 0101xxxxb (e.g., 0x50). The vector math group performs arithmetic such as dot product calculations on vector operands.

[0088] Graphics Pipeline

[0089] FIG. **8** is a block diagram of another embodiment of a graphics processor **800**. 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.

[0090] In some embodiments, graphics processor **800** includes a graphics pipeline **820**, a media pipeline **830**, a display engine **840**, thread execution logic **850**, and a render output pipeline **870**. In some embodiments, graphics processor **800** 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 **800** via a ring interconnect **802**. In some embodiments, ring interconnect **802** couples graphics processor **800** to other processing components, such as other graphics processors or general-purpose processors. Commands from ring interconnect **802** are interpreted by a command streamer **803**, which supplies instructions to individual components of graphics pipeline **820** or media pipeline **830**.

[0091] In some embodiments, command streamer **803** directs the operation of a vertex fetcher **805** that reads vertex data from memory and executes vertex-processing commands provided by command streamer **803**. In some embodiments, vertex fetcher **805** provides vertex data to a vertex shader **807**, which performs coordinate space transformation and lighting operations to each vertex. In some embodiments, vertex fetcher **805** and vertex shader **807** execute vertex-processing instructions by dispatching execution threads to execution units **852A-852B** via a thread dispatcher **831**.

[0092] In some embodiments, execution units **852A-852B** are an array of vector processors having an instruction set for performing graphics and media operations. In some embodi-

8

ments, execution units **852A-852**B have an attached L1 cache **851** that is specific for each array or shared between the arrays. The cache can be configured as a data cache, an instruction cache, or a single cache that is partitioned to contain data and instructions in different partitions.

[0093] In some embodiments, graphics pipeline **820** includes tessellation components to perform hardware-accelerated tessellation of 3D objects. In some embodiments, a programmable hull shader **811** configures the tessellation operations. A programmable domain shader **817** provides back-end evaluation of tessellation output, where tessellator **813** operates at the direction of hull shader **811** and contains special purpose logic to generate a set of detailed geometric objects based on a coarse geometric model that is provided as input to graphics pipeline **820**. In some embodiments, if tessellation is not used, tessellation components (e.g., hull shader **811**, tessellator **813**, and domain shader **817**) can be bypassed.

[0094] In some embodiments, complete geometric objects can be processed by a geometry shader **819** via one or more threads dispatched to execution units **852A-852**B, or can proceed directly to the clipper **829**. 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 **819** receives input from the vertex shader **807**. In some embodiments, geometry shader **819** is programmable by a geometry shader program to perform geometry tessellation if the tessellation units are disabled.

[0095] Before rasterization, a clipper **829** processes vertex data. The clipper **829** may be a fixed function clipper or a programmable clipper having clipping and geometry shader functions. In some embodiments, a rasterizer and depth test component **873** in the render output pipeline **870** 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 **850**. In some embodiments, an application can bypass the rasterizer and depth test component **873** and access un-rasterized vertex data via a stream out unit **823**.

[0096] The graphics processor **800** 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 **852A-852**B and associated cache(s) **851**, texture and media sampler **854**, and texture/sampler cache **858** interconnect via a data port **856** to perform memory access and communicate with render output pipeline components of the processor. In some embodiments, sampler **854**, caches **851**, **858** and execution units **852A-852**B each have separate memory access paths.

[0097] In some embodiments, render output pipeline **870** contains a rasterizer and depth test component **873** 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 **878** and depth cache **879** are also available in some embodiments. A pixel operations component **877** performs pixel-based operations on the data, though in some instances, pixel operations associated with 2D operations (e.g. bit block image transfers with blending) are performed by the 2D engine **841**, or substituted at display time by the display controller **843** using overlay display planes. In some

embodiments, a shared L3 cache **875** is available to all graphics components, allowing the sharing of data without the use of main system memory.

[0098] In some embodiments, graphics processor media pipeline **830** includes a media engine **837** and a video front end **834**. In some embodiments, video front end **834** receives pipeline commands from the command streamer **803**. In some embodiments, media pipeline **830** includes a separate command streamer. In some embodiments, video front-end **834** processes media commands before sending the command to the media engine **837**. In some embodiments, media engine **837** includes thread spawning functionality to spawn threads for dispatch to thread execution logic **850** via thread dispatcher **831**.

[0099] In some embodiments, graphics processor **800** includes a display engine **840**. In some embodiments, display engine **840** is external to processor **800** and couples with the graphics processor via the ring interconnect **802**, or some other interconnect bus or fabric. In some embodiments, display engine **840** includes a 2D engine **841** and a display controller **843**. In some embodiments, display engine **840** contains special purpose logic capable of operating independently of the 3D pipeline. In some embodiments, display controller **843** couples with a display device (not shown), which may be a system integrated display device, as in a laptop computer, or an external display device attached via a display device connector.

[0100] In some embodiments, graphics pipeline **820** and media pipeline **830** are configurable to perform operations based on multiple graphics and media programming interfaces and are not specific to any one application programming interface (API). In 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), Open Computing Language (OpenCL), and/or Vulkan graphics and compute API, all from the Khronos Group. In some embodiments, support may also be provided for the Direct3D library from the Microsoft Corporation. In some embodiments, a combination of these libraries may be supported. 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.

[0101] Graphics Pipeline Programming

[0102] FIG. **9A** is a block diagram illustrating a graphics processor command format **900** according to some embodiments. FIG. **9B** is a block diagram illustrating a graphics processor command sequence **910** according to an embodiment. The solid lined boxes in FIG. **9A** illustrate the components that are generally included in a graphics command while the dashed lines include components that are optional or that are only included in a sub-set of the graphics commands. The exemplary graphics processor command format **900** of FIG. **9A** includes data fields to identify a target client **902** of the command, a command operation code (opcode) **904**, and the relevant data **906** for the command. A sub-opcode **905** and a command size **908** are also included in some commands.

[0103] In some embodiments, client **902** specifies the client unit of the graphics device that processes the com-

9

mand 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 **904** and, if present, sub-opcode **905** to determine the operation to perform. The client unit performs the command using information in data field **906**. For some commands an explicit command size **908** 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.

[0104] The flow diagram in FIG. **9B** shows an exemplary graphics processor command sequence **910**. 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.

[0105] In some embodiments, the graphics processor command sequence **910** may begin with a pipeline flush command **912** to cause any active graphics pipeline to complete the currently pending commands for the pipeline. In some embodiments, the 3D pipeline **922** and the media pipeline **924** do not operate concurrently. The pipeline flush is performed to cause the active graphics pipeline to complete any pending commands. In response to a pipeline flush, the command parser for the graphics processor will pause command processing until the active drawing engines complete pending operations and the relevant read caches are invalidated. Optionally, any data in the render cache that is marked 'dirty' can be flushed to memory. In some embodiments, pipeline flush command **912** can be used for pipeline synchronization or before placing the graphics processor into a low power state.

[0106] In some embodiments, a pipeline select command **913** is used when a command sequence requires the graphics processor to explicitly switch between pipelines. In some embodiments, a pipeline select command **913** is required only once within an execution context before issuing pipeline commands unless the context is to issue commands for both pipelines. In some embodiments, a pipeline flush command **912** is required immediately before a pipeline switch via the pipeline select command **913**.

[0107] In some embodiments, a pipeline control command **914** configures a graphics pipeline for operation and is used to program the 3D pipeline **922** and the media pipeline **924**. In some embodiments, pipeline control command **914** configures the pipeline state for the active pipeline. In one embodiment, the pipeline control command **914** is used for

pipeline synchronization and to clear data from one or more cache memories within the active pipeline before processing a batch of commands.

[0108] In some embodiments, return buffer state commands **916** are used to configure a set of return buffers for the respective pipelines to write data. Some pipeline operations require the allocation, selection, or configuration of one or more return buffers into which the operations write intermediate data during processing. 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 **916** includes selecting the size and number of return buffers to use for a set of pipeline operations.

[0109] The remaining commands in the command sequence differ based on the active pipeline for operations. Based on a pipeline determination **920**, the command sequence is tailored to the 3D pipeline **922** beginning with the 3D pipeline state **930** or the media pipeline **924** beginning at the media pipeline state **940**.

[0110] The commands to configure the 3D pipeline state **930** include 3D state setting commands for vertex buffer state, vertex element state, constant color state, depth buffer state, and other state variables that are to be configured before 3D primitive commands are processed. The values of these commands are determined at least in part based on the particular 3D API in use. In some embodiments, 3D pipeline state **930** commands are also able to selectively disable or bypass certain pipeline elements if those elements will not be used.

[0111] In some embodiments, 3D primitive **932** command is used to submit 3D primitives to be processed by the 3D pipeline. Commands and associated parameters that are passed to the graphics processor via the 3D primitive **932** command are forwarded to the vertex fetch function in the graphics pipeline. The vertex fetch function uses the 3D primitive **932** command data to generate vertex data structures. The vertex data structures are stored in one or more return buffers. In some embodiments, 3D primitive **932** command is used to perform vertex operations on 3D primitives via vertex shaders. To process vertex shaders, 3D pipeline **922** dispatches shader execution threads to graphics processor execution units.

[0112] In some embodiments, 3D pipeline **922** is triggered via an execute **934** 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.

[0113] In some embodiments, the graphics processor command sequence **910** follows the media pipeline **924** path when performing media operations. In general, the specific use and manner of programming for the media pipeline **924** depends on the media or compute operations to be performed. Specific media decode operations may be offloaded to the media pipeline during media decode. 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.

[0114] In some embodiments, media pipeline 924 is configured in a similar manner as the 3D pipeline 922. A set of commands to configure the media pipeline state 940 are dispatched or placed into a command queue before the media object commands 942. In some embodiments, media pipeline state commands 940 include data to configure the media pipeline elements that will be used to process the media objects. This includes data to configure the video decode and video encode logic within the media pipeline, such as encode or decode format. In some embodiments, media pipeline state commands 940 also support the use of one or more pointers to "indirect" state elements that contain a batch of state settings.

[0115] In some embodiments, media object commands 942 supply pointers to media objects for processing by the media pipeline. The media objects include memory buffers containing video data to be processed. In some embodiments, all media pipeline states must be valid before issuing a media object command 942. Once the pipeline state is configured and media object commands 942 are queued, the media pipeline 924 is triggered via an execute command 944 or an equivalent execute event (e.g., register write). Output from media pipeline 924 may then be post processed by operations provided by the 3D pipeline 922 or the media pipeline 924. In some embodiments, GPGPU operations are configured and executed in a similar manner as media operations.

[0116] Graphics Software Architecture

[0117] FIG. 10 illustrates exemplary graphics software architecture for a data processing system 1000 according to some embodiments. In some embodiments, software architecture includes a 3D graphics application 1010, an operating system 1020, and at least one processor 1030. In some embodiments, processor 1030 includes a graphics processor 1032 and one or more general-purpose processor core(s) 1034. The graphics application 1010 and operating system 1020 each execute in the system memory 1050 of the data processing system.

[0118] In some embodiments, 3D graphics application 1010 contains one or more shader programs including shader instructions 1012. The shader language instructions may be in a high-level shader language, such as the High Level Shader Language (HLSL) or the OpenGL Shader Language (GLSL). The application also includes executable instructions 1014 in a machine language suitable for execution by the general-purpose processor core 1034. The application also includes graphics objects 1016 defined by vertex data.

[0119] In some embodiments, operating system 1020 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. The operating system 1020 can support a graphics API 1022 such as the Direct3D API, the OpenGL API, or the Vulkan API. When the Direct3D API is in use, the operating system 1020 uses a front-end shader compiler 1024 to compile any shader instructions 1012 in HLSL into a lower-level shader language. The compilation may be a just-in-time (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 1010. In some embodiments, the shader instructions 1012 are provided in an intermediate form, such as a version of the Standard Portable Intermediate Representation (SPIR) used by the Vulkan API.

[0120] In some embodiments, user mode graphics driver 1026 contains a back-end shader compiler 1027 to convert the shader instructions 1012 into a hardware specific representation. When the OpenGL API is in use, shader instructions 1012 in the GLSL high-level language are passed to a user mode graphics driver 1026 for compilation. In some embodiments, user mode graphics driver 1026 uses operating system kernel mode functions 1028 to communicate with a kernel mode graphics driver 1029. In some embodiments, kernel mode graphics driver 1029 communicates with graphics processor 1032 to dispatch commands and instructions.

[0121] IP Core Implementations

[0122] 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.

[0123] FIG. 11 is a block diagram illustrating an IP core development system 1100 that may be used to manufacture an integrated circuit to perform operations according to an embodiment. The IP core development system 1100 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 1130 can generate a software simulation 1110 of an IP core design in a high level programming language (e.g., C/C++). The software simulation 1110 can be used to design, test, and verify the behavior of the IP core using a simulation model 1112. The simulation model 1112 may include functional, behavioral, and/or timing simulations. A register transfer level (RTL) design 1115 can then be created or synthesized from the simulation model 1112. The RTL design 1115 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 1115, 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.

[0124] The RTL design **1115** or equivalent may be further synthesized by the design facility into a hardware model **1120**, 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 **1165** using non-volatile memory **1140** (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 **1150** or wireless connection **1160**. The fabrication facility **1165** 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.

[0125] Exemplary System on a Chip Integrated Circuit

[0126] FIGS. **12-14** illustrated exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included, including additional graphics processors/cores, peripheral interface controllers, or general purpose processor cores.

[0127] FIG. **12** is a block diagram illustrating an exemplary system on a chip integrated circuit **1200** that may be fabricated using one or more IP cores, according to an embodiment. Exemplary integrated circuit **1200** includes one or more application processor(s) **1205** (e.g., CPUs), at least one graphics processor **1210**, and may additionally include an image processor **1215** and/or a video processor **1220**, any of which may be a modular IP core from the same or multiple different design facilities. Integrated circuit **1200** includes peripheral or bus logic including a USB controller **1225**, UART controller **1230**, an SPI/SDIO controller **1235**, and an I²S/I²C controller **1240**. Additionally, the integrated circuit can include a display device **1245** coupled to one or more of a high-definition multimedia interface (HDMI) controller **1250** and a mobile industry processor interface (MIPI) display interface **1255**. Storage may be provided by a flash memory subsystem **1260** including flash memory and a flash memory controller. Memory interface may be provided via a memory controller **1265** for access to SDRAM or SRAM memory devices. Some integrated circuits additionally include an embedded security engine **1270**.

[0128] FIG. **13** is a block diagram illustrating an exemplary graphics processor **1310** of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment. Graphics processor **1310** can be a variant of the graphics processor **1210** of FIG. **12**. Graphics processor **1310** includes a vertex processor **1305** and one or more fragment processor(s) **1315A1315N** (e.g., **1315A**, **1315B**, **1315C**, **1315D**, through **1315N-1**, and **1315N**). Graphics processor **1310** can execute different shader programs via separate logic, such that the vertex processor **1305** is optimized to execute operations for vertex shader programs, while the one or more fragment processor(s) **1315A-1315N** execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. The vertex processor **1305** performs the vertex processing stage of the 3D graphics pipeline and generates primitives and vertex data. The fragment processor(s) **1315A-1315N** use the primitive and vertex data generated by the vertex processor **1305** to produce a framebuffer that is displayed on a display device. In one embodiment, the fragment processor(s) **1315A-1315N** are optimized to execute fragment shader programs as provided for in the OpenGL API, which may be used to perform similar operations as a pixel shader program as provided for in the Direct 3D API.

[0129] Graphics processor **1310** additionally includes one or more memory management units (MMUs) **1320A-1320B**, cache(s) **1325A-1325B**, and circuit interconnect(s) **1330A-1330B**. The one or more MMU(s) **1320A-1320B** provide for virtual to physical address mapping for integrated circuit **1310**, including for the vertex processor **1305** and/or fragment processor(s) **1315A-1315N**, which may reference vertex or image/texture data stored in memory, in addition to vertex or image/texture data stored in the one or more cache(s) **1325A-1325B**. In one embodiment the one or more MMU(s) **1325A-1325B** may be synchronized with other MMUs within the system, including one or more MMUs associated with the one or more application processor(s) **1205**, image processor **1215**, and/or video processor **1220** of FIG. **12**, such that each processor **1205-1220** can participate in a shared or unified virtual memory system. The one or more circuit interconnect(s) **1330A-1330B** enable graphics processor **1310** to interface with other IP cores within the SoC, either via an internal bus of the SoC or via a direct connection, according to embodiments.

[0130] FIG. **14** is a block diagram illustrating an additional exemplary graphics processor **1410** of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment. Graphics processor **1410** can be a variant of the graphics processor **1210** of FIG. **12**. Graphics processor **1410** includes the one or more MMU(s) **1320A-1320B**, caches **1325A-1325B**, and circuit interconnects **1330A-1330B** of the integrated circuit **1300** of FIG. **13**.

[0131] Graphics processor **1410** includes one or more shader core(s) **1415A-1415N** (e.g., **1415A**, **1415B**, **1415C**, **1415D**, **1415E**, **1415F**, through **1315N-1**, and **1315N**), which provides for a unified shader core architecture in which a single core or type or core can execute all types of programmable shader code, including shader program code to implement vertex shaders, fragment shaders, and/or compute shaders. The exact number of shader cores present can vary among embodiments and implementations. Additionally, graphics processor **1410** includes an inter-core task manager **1405**, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores **1415A-1415N** and a tiling unit **1418** to accelerate tiling operations for tile-based rendering, in which rendering operations for a scene are subdivided in image space, for example to exploit local spatial coherence within a scene or to optimize use of internal caches.

[0132] FIG. **15** illustrates a computing device **1500** employing a multi-resolution deferred shading mechanism ("shading mechanism") **1510** according to one embodiment. Computing device **1500** (e.g., smart wearable devices, virtual reality (VR) devices, head-mounted display (HMDs), mobile computers, Internet of Things (IoT) devices, laptop computers, desktop computers, server computers, etc.) may be the same as data processing system **100** of FIG. **1** and accordingly, for brevity, clarity, and ease of understanding, many of the details stated above with reference to FIGS. **1-14** are not further discussed or repeated hereafter. As illustrated, in one embodiment, computing device **1500** is shown as hosting shading mechanism **1510**.

[0133] As illustrated, in one embodiment, shading mechanism **1510** may be hosted by or part of graphics driver **1516**. In another embodiment, shading mechanism **1510** may be hosted by or part of firmware of graphics processing unit ("GPU" or "graphics processor") **1514**. In yet another embodiment, shading mechanism **1510** may be hosted by or part of firmware of central processing unit ("CPU" or "application processor") **1512**.

[0134] In yet another embodiment, shading mechanism **1510** may be hosted as software or firmware logic by operating system **1506**. In yet another embodiment, shading mechanism **1510** may be partially and simultaneously hosted by multiple components of computing device **1500**, such as one or more of graphics driver **1516**, GPU **1514**, GPU firmware, CPU **1512**, CPU firmware, operating system **1506**, and/or the like. For brevity, clarity, and ease of understanding, throughout the rest of this document, shading mechanism **1510** is shown and discussed as part of GPU **1514**; however, embodiments are not limited as such. It is contemplated and to be noted that shading mechanism **1510** or one or more of its components may be implemented as hardware, software, and/or firmware.

[0135] Throughout the document, term "user" may be interchangeably referred to as "viewer", "observer", "person", "individual", "end-user", and/or the like. It is to be noted that throughout this document, terms like "graphics domain" may be referenced interchangeably with "graphics processing unit", "graphics processor", or simply "GPU" and similarly, "CPU domain" or "host domain" may be referenced interchangeably with "computer processing unit", "application processor", or simply "CPU".

[0136] Computing device **1500** may include any number and type of communication devices, such as large computing systems, such as server computers, desktop computers, etc., and may further include set-top boxes (e.g., Internet-based cable television set-top boxes, etc.), global positioning system (GPS)-based devices, etc. Computing device **1500** may include mobile computing devices serving as communication devices, such as cellular phones including smartphones, personal digital assistants (PDAs), tablet computers, laptop computers, e-readers, smart televisions, television platforms, wearable devices (e.g., glasses, watches, bracelets, smartcards, jewelry, clothing items, etc.), media players, etc. For example, in one embodiment, computing device **1500** may include a mobile computing device employing a computer platform hosting an integrated circuit ("IC"), such as system on a chip ("SoC" or "SOC"), integrating various hardware and/or software components of computing device **1500** on a single chip.

[0137] As illustrated, in one embodiment, computing device **1500** may include any number and type of hardware and/or software components, such as (without limitation) GPU **1514**, graphics driver (also referred to as "GPU driver", "graphics driver logic", "driver logic", user-mode driver (UMD), UMD, user-mode driver framework (UMDF), UMDF, or simply "driver") **1516**, CPU **1512**, memory **1508**, network devices, drivers, or the like, as well as input/output (I/O) sources **1504**, such as touchscreens, touch panels, touch pads, virtual or regular keyboards, virtual or regular mice, ports, connectors, etc. Computing device **1500** may include operating system (OS) **1506** serving as an interface between hardware and/or physical resources of the computer device **1500** and a user. It is contemplated that CPU **1512** may include one or more

processors, such as processor(s) **102** of FIG. **1**, while GPU **1514** may include one or more graphics processors, such as graphics processor(s) **108** of FIG. **1**.

[0138] It is to be noted that terms like "node", "computing node", "server", "server device", "cloud computer", "cloud server", "cloud server computer", "machine", "host machine", "device", "computing device", "computer", "computing system", and the like, may be used interchangeably throughout this document. It is to be further noted that terms like "application", "software application", "program", "software program", "package", "software package", and the like, may be used interchangeably throughout this document. Also, terms like "job", "input", "request", "message", and the like, may be used interchangeably throughout this document.

[0139] It is contemplated and as further described with reference to FIGS. **1-14**, some processes of the graphics pipeline as described above are implemented in software, while the rest are implemented in hardware. A graphics pipeline may be implemented in a graphics coprocessor design, where CPU **1512** is designed to work with GPU **1514** which may be included in or co-located with CPU **1512**. In one embodiment, GPU **1514** may employ any number and type of conventional software and hardware logic to perform the conventional functions relating to graphics rendering as well as novel software and hardware logic to execute any number and type of instructions, such as instructions **121** of FIG. **1**, to perform the various novel functions of shading mechanism **1510** as disclosed throughout this document.

[0140] As aforementioned, memory **1508** may include a random access memory (RAM) comprising application database having object information. A memory controller hub, such as memory controller hub **116** of FIG. **1**, may access data in the RAM and forward it to GPU **1514** for graphics pipeline processing. RAM may include double data rate RAM (DDR RAM), extended data output RAM (EDO RAM), etc. CPU **1512** interacts with a hardware graphics pipeline, as illustrated with reference to FIG. **3**, to share graphics pipelining functionality. Processed data is stored in a buffer in the hardware graphics pipeline, and state information is stored in memory **1508**. The resulting image is then transferred to I/O sources **1504**, such as a display component, such as display device **320** of FIG. **3**, for displaying of the image. It is contemplated that the display device may be of various types, such as Cathode Ray Tube (CRT), Thin Film Transistor (TFT), Liquid Crystal Display (LCD), Organic Light Emitting Diode (OLED) array, etc., to display information to a user.

[0141] Memory **1508** may comprise a pre-allocated region of a buffer (e.g., frame buffer); however, it should be understood by one of ordinary skill in the art that the embodiments are not so limited, and that any memory accessible to the lower graphics pipeline may be used. Computing device **1500** may further include input/output (I/O) control hub (ICH) **150** as referenced in FIG. **1**, one or more I/O sources **1504**, etc.

[0142] CPU **1512** may include one or more processors to execute instructions in order to perform whatever software routines the computing system implements. The instructions frequently involve some sort of operation performed upon data. Both data and instructions may be stored in system memory **1508** and any associated cache. Cache is typically designed to have shorter latency times than system memory

1508; for example, cache might be integrated onto the same silicon chip(s) as the processor(s) and/or constructed with faster static RAM (SRAM) cells whilst the system memory 1508 might be constructed with slower dynamic RAM (DRAM) cells. By tending to store more frequently used instructions and data in the cache as opposed to the system memory 1508, the overall performance efficiency of computing device 1500 improves. It is contemplated that in some embodiments, GPU 1514 may exist as part of CPU 1512 (such as part of a physical CPU package) in which case, memory 1508 may be shared by CPU 1512 and GPU 1514 or kept separated.

[0143] System memory 1508 may be made available to other components within the computing device 1500. For example, any data (e.g., input graphics data) received from various interfaces to the computing device 1500 (e.g., keyboard and mouse, printer port, Local Area Network (LAN) port, modem port, etc.) or retrieved from an internal storage element of the computer device 1500 (e.g., hard disk drive) are often temporarily queued into system memory 1508 prior to their being operated upon by the one or more processor(s) in the implementation of a software program. Similarly, data that a software program determines should be sent from the computing device 1500 to an outside entity through one of the computing system interfaces, or stored into an internal storage element, is often temporarily queued in system memory 1508 prior to its being transmitted or stored.

[0144] Further, for example, an ICH, such as ICH 130 of FIG. 1, may be used for ensuring that such data is properly passed between the system memory 1508 and its appropriate corresponding computing system interface (and internal storage device if the computing system is so designed) and may have bi-directional point-to-point links between itself and the observed I/O sources/devices 1504. Similarly, an MCH, such as MCH 116 of FIG. 1, may be used for managing the various contending requests for system memory 1508 accesses amongst CPU 1512 and GPU 1514, interfaces and internal storage elements that may proximately arise in time with respect to one another.

[0145] I/O sources 1504 may include one or more 110 devices that are implemented for transferring data to and/or from computing device 1500 (e.g., a networking adapter); or, for a large scale non-volatile storage within computing device 1500 (e.g., hard disk drive). User input device, including alphanumeric and other keys, may be used to communicate information and command selections to GPU 1514. Another type of user input device is cursor control, such as a mouse, a trackball, a touchscreen, a touchpad, or cursor direction keys to communicate direction information and command selections to GPU 1514 and to control cursor movement on the display device. Camera and microphone arrays of computer device 1500 may be employed to observe gestures, record audio and video and to receive and transmit visual and audio commands.

[0146] Computing device 1500 may further include network interface(s) to provide access to a network, such as a LAN, a wide area network (WAN), a metropolitan area network (MAN), a personal area network (PAN), Bluetooth, a cloud network, a mobile network (e.g., 3rd Generation (3G), 4th Generation (4G), etc.), an intranet, the Internet, etc. Network interface(s) may include, for example, a wireless network interface having antenna, which may represent one or more antenna(e). Network interface(s) may also include,

for example, a wired network interface to communicate with remote devices via network cable, which may be, for example, an Ethernet cable, a coaxial cable, a fiber optic cable, a serial cable, or a parallel cable.

[0147] Network interface(s) may provide access to a LAN, for example, by conforming to IEEE 802.11b and/or IEEE 802.11g standards, and/or the wireless network interface may provide access to a personal area network, for example, by conforming to Bluetooth standards. Other wireless network interfaces and/or protocols, including previous and subsequent versions of the standards, may also be supported. In addition to, or instead of, communication via the wireless LAN standards, network interface(s) may provide wireless communication using, for example, Time Division, Multiple Access (TDMA) protocols, Global Systems for Mobile Communications (GSM) protocols, Code Division, Multiple Access (CDMA) protocols, and/or any other type of wireless communications protocols.

[0148] Network interface(s) may include one or more communication interfaces, such as a modem, a network interface card, or other well-known interface devices, such as those used for coupling to the Ethernet, token ring, or other types of physical wired or wireless attachments for purposes of providing a communication link to support a LAN or a WAN, for example. In this manner, the computer system may also be coupled to a number of peripheral devices, clients, control surfaces, consoles, or servers via a conventional network infrastructure, including an Intranet or the Internet, for example.

[0149] It is to be appreciated that a lesser or more equipped system than the example described above may be preferred for certain implementations. Therefore, the configuration of computing device 1500 may vary from implementation to implementation depending upon numerous factors, such as price constraints, performance requirements, technological improvements, or other circumstances. Examples of the electronic device or computer system 1500 may include (without limitation) a mobile device, a personal digital assistant, a mobile computing device, a smartphone, a cellular telephone, a handset, a one-way pager, a two-way pager, a messaging device, a computer, a personal computer (PC), a desktop computer, a laptop computer, a notebook computer, a handheld computer, a tablet computer, a server, a server array or server farm, a web server, a network server, an Internet server, a work station, a mini-computer, a main frame computer, a supercomputer, a network appliance, a web appliance, a distributed computing system, multiprocessor systems, processor-based systems, consumer electronics, programmable consumer electronics, television, digital television, set top box, wireless access point, base station, subscriber station, mobile subscriber center, radio network controller, router, hub, gateway, bridge, switch, machine, or combinations thereof.

[0150] Embodiments may be implemented as any or a combination of: one or more microchips or integrated circuits interconnected using a parentboard, hardwired logic, software stored by a memory device and executed by a microprocessor, firmware, an application specific integrated circuit (ASIC), and/or a field programmable gate array (FPGA). The term "logic" may include, by way of example, software or hardware and/or combinations of software and hardware.

[0151] Embodiments may be provided, for example, as a computer program product which may include one or more

14

machine-readable media having stored thereon machine-executable instructions that, when executed by one or more machines such as a computer, network of computers, or other electronic devices, may result in the one or more machines carrying out operations in accordance with embodiments described herein. A machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs, RAMs, EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electrically Erasable Programmable Read Only Memories), magnetic or optical cards, flash memory, or other type of media/machine-readable medium suitable for storing machine-executable instructions.

[0152] Moreover, embodiments 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 one or more data signals embodied in and/or modulated by a carrier wave or other propagation medium via a communication link (e.g., a modem and/or network connection).

[0153] FIG. 16 illustrates shading mechanism 1510 of FIG. 15 according to one embodiment. For brevity, many of the details already discussed with reference to FIGS. 1-15 are not repeated or discussed hereafter. In one embodiment, shading mechanism 1510 may include any number and type of components, such as (without limitations): detection/reception logic 1601; division/splitting logic 1605; pixel shading computation logic ("computation logic") 1607; shading pass logic 1609; execution/forwarding logic 1611; and communication/compatibility logic 1613.

[0154] Computing device 1500 is further shown to be in communication with one or more repositories, datasets, and/or databases, such as database(s) 1630 (e.g., cloud storage, non-cloud storage, etc.), where database(s) 1630 may reside at a local storage or a remote storage over communication medium(s) 1625, such as one or more networks (e.g., cloud network, proximity network, mobile network, intranet, Internet, etc.).

[0155] It is contemplated that a software application running at computing device 1500 may be responsible for performing or facilitating performance of any number and type of tasks using one or more components (e.g., GPU 1514, graphics driver 1516, CPU 1512, etc.) of computing device 1500. When performing such tasks, as defined by the software application, one or more components, such as GPU 1514, graphics driver 1516, CPU 1512, etc., may communicate with each other to ensure accurate and timely processing and completion of those tasks.

[0156] Prior to discussing the workings of shading mechanism 1510, let us consider that conventional deferred shading is computed per-pixel. Applications typically compute acceleration information, such as per-tile light lists of all light sources that contribute to a tile of pixels. Then, for each light source and pixel in the image, expensive lighting computations are performed. The cost associated with per-pixel deferred shading is particularly prohibitive when the number of pixels gets very large and thus, to reduce pixel count, several conventional techniques are employed.

[0157] One conventional technique, known as uniformly reduced resolution, is intended to be used to reduce both shading and memory bandwidth, but, in turn, significantly reduces image quality. To improve quality smart upscaling filters (e.g., bilateral) are commonly used, but features that are missed due to under-sampling are still lost, which shows up as jaggies along edges and/or flickering geometric aliasing.

[0158] Another conventional technique, known as multi-pass deferred shading, allows for multiple fullscreen passes, where each shading is a subset of the pixels in a geometry buffer (also referred to as "G-buffer"), such that the result is blended/filtered together in yet another pass. Applications can use a stencil buffer to select between per-pixel shading and reduced rate shading (e.g., once per 2×2 pixels), but the drawback is consumption of additional memory bandwidth and the dealing with an overhead of performing multiple fullscreen passes. Further, depending on how the data is stored, using hardware texture filtering in the blending/filtering pass is also impractical.

[0159] Another conventional technique, known as deferred shading using coarse pixel shading (CPS), is used to select between two different rates, such as coarse pixel and pixel rate, where a coarse pixel size can be of any of [1,2,4]×[1,2,4] pixels, but remains constant per draw call or determined based on a radial falloff. A drawback of this conventional technique is that the rate cannot be chosen at a fine granularity and that the final image is not smoothly filtered. Further, any performance benefits may be small or none, unless all pixels under a coarse pixel are discarded; otherwise, the shader needs to run at both rates.

[0160] Embodiments provide for novel technique, through shading mechanism 1510, to facilitate 1) performance of shading in a single pass, 2) controlling of shading granularity at a relatively fine granularity, and 3) smoothly filtering of the shading, where these factors allow for improvement of performance and/or image quality.

[0161] In one embodiment, shading mechanism 1510 utilizes the texel shader (TS) stage of a graphics pipeline, where it is contemplated that the TS stage is expected to be part of current and future 3D graphics APIs. For example, an exemplary 3D graphics pipeline having a TS stage is shown with reference to FIG. 17A and as illustrated, a TS is triggered by a pixel shader (PS), where the PS requests texel shading by issuing "Evaluate" operations on a PT, where the PT is a sparsely populated texture, where each texel can be either "unshaded" or "shaded", where the TS is invoked the first time an "unshaded" texel is accessed. As illustrated in FIG. 17A, in this case, the output of the TS is written to the PT and the texel is marked as "shaded".

[0162] The Evaluate operation ensures that all texels that lie under the texture filter footprint are shaded. The footprint is determined by the sampling mode and texture coordinates (u,v). Note that procedural textures may be a mipmap hierarchy. Thus, a single Evaluate operation can trigger texel shaders for multiple texels in one or for one or more mip map levels and multiple texels in each mip.

[0163] It is contemplated that embodiments are not limited to any particular graphics pipeline or its various stages and their variants. For example, in some embodiments, filtered shading is immediately returned to the calling PS, while, in some other embodiments, the resulting PT is sampled in a later rendering pass.

[0164] In one embodiment, shading mechanism 1510 provides for moving the deferred lighting computations into a TS, where the TS operates on a fullscreen PT with multiple mipmap levels, such as the base resolution of the PT is the same as the screen dimensions, while any subsequent mip levels are smaller. Further, in one embodiment, shading

mechanism **1510** allows for splitting the conventional fullscreen deferred shading pass into two shaders: 1) a pixel shader that determines shading rate and triggers texel shading; and 2) texel shader that performs the actual lighting computations.

[0165] In one embodiment, detection/reception logic **1601** may be used to detect a relevant stage in a graphics pipeline, such a whether a particular pipeline stage has just commenced or ended or is still being performed. For example, detection/reception logic **1601** may keep detect and keep track of any number and type of graphics pipeline stages, such as rendering G-buffer, computing per-tile, light lists, etc., so that they may be communicated with other components, such as division/splitting logic **1605**, computation logic **1607**, shading pass logic **1609**, etc., to perform their tasks to achieve efficiency and higher performance.

[0166] For example, upon reaching a fullscreen deferred shading pass, division/splitting logic **1605** may be triggered to divide or split the process into being performed by two shaders, such as a pixel shader and a texel shader as represented by computation logic **1607** and shading pass logic **1609**, respectively. For example, computation logic **1607** may be used to determine or compute the shading rate and trigger shading pass logic **1609**, while shading pass logic **1609** to perform any lighting computations. For example, the shading rate, as computed by computation logic **1607**, may be effectively controlled by, first, selecting the mip level at which the TS executes, where this selection can be performed at a fine granularity, such as per pixel, allowing an application to shade finely near important features and more coarsely in other areas of the screen.

[0167] It is contemplated that shader at coarse level (e.g., higher mip) may translate into each texel covering a larger screen space area. For example, shading at lod=2 using a mipmap chain with resolution, as per below, may reflect that each texel occupies 4×4 pixels in screen space: lod 0: 1920×1080 (screen resolution); lod 1: 960×540 (1/2 resolution); 3) lod 2: 480×270 (1/4 resolution); and lod 3: 240×135 (1/8 resolution).

[0168] In one embodiment, as illustrated with respect to FIG. **18**A, shading rate may be determined or computed by computation logic **1607** in a separate fullscreen pass prior to the main lighting pass. In another embodiment, FIG. **18**B illustrates a transaction sequence that assumes a 2-pass texel shader solution, where texel shading is first triggered and then sampled in a separate fullscreen pass (such as through fetch texel shading) as facilitated by shading pass logic **1609**. Stated differently, in one embodiment, shading rate is computed in a separate pass before the fullscreen shading pass, while, in another embodiment, this shading rate is computed as part of the fullscreen shading pass immediately before triggering the TS. It is contemplated that transparent surfaces may be handled the same way as is done in other techniques, such as by blending transparent (e.g., alpha) objects on top of the final frame buffer after deferred shading.

[0169] In one embodiment, once the per-pixel shading rate is computed, as facilitated by computation logic **1607**, and a fullscreen shading pass (e.g., texture space) is performed, as facilitated by shading pass logic **1609**, execution/forwarding logic **1611** may be used to execute these tasks and perform any application of such tasks in moving the process forward in the graphics pipeline, such as by using the shading rate in performing a fullscreen shading pass, fol-lowed by triggering or initiating a next stage, such as a fullscreen post-processing pass, in the graphics pipeline.

[0170] In one embodiment, if the shading rate is computed, as facilitated by computation logic **1607**, in a separate pass prior to the main lighting/shading pass, as facilitated by shading pass logic **1609**, the shading may then be stored in a compact fullscreen texture to minimize memory bandwidth, such as using an 8 bits/pixel format (e.g. DXGI_FORMAT_R8_UNORM in DirectX®). In some embodiments, the shading rate is not computed per pixel, but rather per group of pixels (such as a tile).

[0171] It is contemplated that computation of the desired shading rate can be performed using a number of different heuristics. For example, it may be desirable to generally shade at a high rate (e.g., full per-pixel rate) near sharp discontinuities to avoid blurring across visible edges in the image. For example, it may be desirable to shade at the full rate near the silhouettes objects. With regard to computing games, similar strategies may be used in determining where to apply anti-aliasing, etc.

[0172] Examples of suitable heuristics for computing the shading rate may include, but are not limited to, one or more of the following metrics: 1) scene-based heuristics, such as (without limitation) a) depth discontinuities, b) normal difference exceeding some threshold, c) surface curvature based on normal, d) object/primitive/texture identification, e) color difference in diffuse surface texture, f) texture coordinates; g) depth-of-field (e.g., depth from focus plane), and h) motion vectors; 2) virtual/augmented reality, such as (without limitation) a) per-pixel bias based on lens/rendering distortion, and b) reduced shading rate in the periphery; and 3) external/contextual input, such as (without limitation) a) eye-tracking information, and b) gaze point prediction based on content.

[0173] With regard to filtering and over-shading, depending on the filtering mode used to trigger texel shading, as facilitated by shading pass logic **1609**, multiple mipmap levels may be shaded. For example, if using trilinear filtering and a fractional shading rate (e.g., lod=1.5), both mip level 1 and level 2 may be shaded for a group of texels and the shading linearly interpolated between the 8 nearest texels (e.g., 2×2 at each mip level). When sampling the resulting shading from the PT, the application may get a smooth transition from per-pixel shading to the lower shading rates.

[0174] Similarly, it is contemplated that the hardware may generally shade groups of multiple texels together, where the minimum granularity may, for example, be 4×4 texels to fill a single instruction, multiple data (SIMD) 16 dispatch. Using trilinear filtering, where two mip levels may be shaded, this can result in more over-shading and smaller performance gains, compared to using bilinear or point sampling.

[0175] With regard to pseudo code, an exemplary pseudo code for conventional per-pixel deferred shading may be as follows:

```
// Traditional per-pixel deferred shading
// -------------------------------------
Texture2D<float3>  position  : register(t0)
Texture2D<float3>  normal    : register(t1)
Texture2D<float2>  texcoord  : register(t2)
float4 MainPS(float4 pos : SV_Position) : SV_Target
{
    // Fetch attributes
```

16

-continued

```
        float3  P = position.Load(int3(pos.xy, 0));
        float3  N = normal.Load(int3(pos.xy, 0));
        float2  uv = texcoord.Load(int3(pos.xy, 0));
        // Compute shading
        return ComputeLighting(P, N, uv);
    }
```

[0176] In one embodiment, a novel pseudo code for multi-resolution deferred shading, as facilitated by shading mechanism **1510**, may appear as follows:

```
// Multi-resolution deferred shading
// -------------------------------------
Texture2D<float3>  position  : register(t0)
Texture2D<float3>  normal   : register(t1)
Texture2D<float2>  texcoord  : register(t2)
Texture2D<float>   shadingLOD            : register(t3)
ProcTexture2D<float4> procTex            : register(p0)
void MainPS(float4 pos : SV_Position)
{
    float lod = shadingLOD.Load(int3(pos.xy,0));
    float2 uv = pos.xy * one_over_frame_size;
    // Trigger texel shader
    procTex.EvaluateLevel(samplerLinear, uv, lod);
}
float4 MainTS(float4 pos : SV_TexelPosFloat)
    : SV_ProcTex
{
    // Fetch attributes
    float3 P = position.Sample(sampLinear, pos.xy);
    float3 N = normal.Sample(sampLinear, pos.xy);
    float3 uv = texcoord.Sample(sampLinear, pos.xy);
    // Compute shading
    return ComputeLighting(P, N, uv);
}
```

[0177] For example, as illustrated above, the pixel shader (MainPS) in the illustrated embodiment of the novel code may be responsible for one or more of: 1) loading/computing the shading level-of-detail, and 2) triggering the texel shader based on the pixel position and the lod. The illustrated embodiment of the code may assume a 2-pass texel shader implementation, where the evaluate level (EvaluateLevel) function triggers texel shading, but does not return a result. Similarly, the texel shader (MainTS) in the illustrated embodiment of the novel code starts by fetching G-buffer attributes. As illustrated, here, bilinear sampling is used to access the G-buffer, as the texel shader may be running at a lower resolution. There may not always be a 1:1 correspondence between texels and pixels and by bilinearly sampling the attributes, we get smooth transitions.

[0178] In one embodiment, such attributes may be directly loaded from the nearest pixel (point sampling) for slightly improved performance at the expensive of image quality. In some embodiments, smarter filters may be employed to fetch G-buffer attributes, for example, bilateral filters may avoid the risk of blurring across edges. After such attributes are fetched, the texel shader may perform the lighting or shading computations (ComputeLighting) identical to traditional deferred shading. This is where the bulk of the work lies and performance gains may come from the fact that often the majority of the images may be shaded at significantly lower than pixel rate.

[0179] In some embodiments, main shading function may remain unmodified, while other functions, such as how attributes are fetched, may be modified. Further, in some embodiments, shaders for per-pixel deferred shading are automatically or semi-automatically transformed into PS/TS pairs for multi-resolution deferred shading, where such code transformations may be implemented as part of the shader compiler, as part of API utility libraries, or as part of third-party tools. For example, in some embodiments, a fullscreen PT may be allocated, where this is of the same dimensions as the frame buffer, so such an allocation may be performed at the time of creating the frame buffer.

[0180] Communication/compatibility logic **1613** may be used to facilitate dynamic communication and compatibility between computing device **1500** and any number and type of other computing devices (such as mobile computing device, desktop computer, server computing device, etc.); processing devices or components (such as CPUs, GPUs, etc.); capturing/sensing/detecting devices (such as capturing/sensing components including cameras, depth sensing cameras, camera sensors, red green blue (RGB) sensors, microphones, etc.); display devices (such as output components including display screens, display areas, display projectors, etc.); user/context-awareness components and/or identification/verification sensors/devices (such as biometric sensors/detectors, scanners, etc.); database(s) **1630**, such as memory or storage devices, databases, and/or data sources (such as data storage devices, hard drives, solid-state drives, hard disks, memory cards or devices, memory circuits, etc.); communication medium(s) **1625**, such as one or more communication channels or networks (e.g., Cloud network, the Internet, intranet, cellular network, proximity networks, such as Bluetooth, Bluetooth low energy (BLE), Bluetooth Smart, Wi-Fi proximity, Radio Frequency Identification (RFID), Near Field Communication (NFC), Body Area Network (BAN), etc.); wireless or wired communications and relevant protocols (e.g., Wi-Fi®, WiMAX, Ethernet, etc.); connectivity and location management techniques; software applications/websites (e.g., social and/or business networking websites, etc., business applications, games and other entertainment applications, etc.); and programming languages, etc., while ensuring compatibility with changing technologies, parameters, protocols, standards, etc.

[0181] Throughout this document, terms like "logic", "component", "module", "framework", "engine", "mechanism", and the like, may be referenced interchangeably and include, by way of example, software, hardware, and/or any combination of software and hardware, such as firmware. In one example, "logic" may refer to or include a software component that is capable of working with one or more of an operating system (e.g., operating system **1506**), a graphics driver (e.g., graphics driver **1516**), etc., of a computing device, such as computing device **1500**. In another example, "logic" may refer to or include a hardware component that is capable of being physically installed along with or as part of one or more system hardware elements, such as an application processor (e.g., CPU **1512**), a graphics processor (e.g., GPU **1514**), etc., of a computing device, such as computing device **1500**. In yet another embodiment, "logic" may refer to or include a firmware component that is capable of being part of system firmware, such as firmware of an application processor (e.g., CPU **1512**) or a graphics processor (e.g., GPU **1514**), etc., of a computing device, such as computing device **1500**.

[0182] Further, any use of a particular brand, word, term, phrase, name, and/or acronym, such as "GPU", "GPU domain", "GPGPU", "CPU", "CPU domain", "graphics

driver", "workload", "application", "graphics pipeline", "pipeline processes", "pixel shader", "PS", "texel shader", "TS", "per-tile shading rate", "shading rate", "fullscreen shading pass", "fetch shading", "execution unit", "EU", "instruction", "API", "3D API", "OpenGL®", "DirectX®", "fragment shader", "YUV texture", "shader execution", "existing UAV capabilities", "existing backend", "hardware", "software", "agent", "graphics driver", "kernel mode graphics driver", "user-mode driver", "user-mode driver framework", "buffer", "graphics buffer", "task", "process", "operation", "software application", "game", etc., should not be read to limit embodiments to software or devices that carry that label in products or in literature external to this document.

[0183] It is contemplated that any number and type of components may be added to and/or removed from shading mechanism **1510** to facilitate various embodiments including adding, removing, and/or enhancing certain features. For brevity, clarity, and ease of understanding of shading mechanism **1510**, many of the standard and/or known components, such as those of a computing device, are not shown or discussed here. It is contemplated that embodiments, as described herein, are not limited to any particular technology, topology, system, architecture, and/or standard and are dynamic enough to adopt and adapt to any future changes.

[0184] FIG. **17**A illustrates a conventional graphics pipeline **1700** implementing texel shading. For brevity, many of the details previously discussed with reference to FIGS. **1-16** may not be discussed or repeated hereafter. As illustrated, input assembler (IA) **1701** reads index and vertex data from memory and feeds vertex shader (VS) **1703**, where VS **1703** to perform shading operations on each vertex, such as changing each vertex's three-dimensional (3D) position in virtual space to a three-dimensional homogeneous (3DH) position in clip space, which after perspective division defines the two-dimensional (2D) coordinate at which it appears on the screen and generates results in the form of primitives (e.g., triangles). This is then followed by a tessellation portion of graphics pipeline **1700**, where the tessellation portion includes hull shader (HS) **1705** and domain shader (DS) **1707** to compute high-detailed surface geometry based on lower-detail input surface geometry that is defined as triangles or quads, etc. For example, the lower-detail input primitives may be called patches. For example, HS **1705** accepts an input patch and produces an output patch or patches that are then used to produced tiled, normalized domains of appropriate type, such as quads or triangles, while DS **1707** is executed against these domains to compute the actual vertex position of any given point in a domain resulting from tessellation, outputting vertex positions.

[0185] Continuing with pipeline **1700**, geometry shader (GS) **1709** takes inputs, such as a primitive, a vertex position, etc., producing corresponding outputs, such as more primitives, etc. GS **1709** is more like a hull/domain shader which takes input vertices and produces output vertices such that for a single input vertex, several output vertices can be produced so that they can be used to generate geometry.

[0186] As illustrated, pixel shader (PS) **1711** then issues evaluate request **1721**, such as evaluate texels shading request, on procedural texture (PT) **1719**. The issuing thread may continue its execution, passing shaded pixels to the output merger (OM) **1713** to perform various operations,

such as alpha blending, and writes the pixels back to the backbuffer, producing render target **1715**. Any un-shaded texels are marked as "shaded" and one or more threads associated with TS **1717** are scheduled to evaluate their shading and write the results (e.g., colors) to memory within PT **1719**.

[0187] FIG. **17**B illustrates a conventional method **1730** for deferred shading. For brevity, many of the details previously discussed with reference to FIGS. **1-17**A may not be discussed or repeated hereafter. As illustrated, method **1730** starts at block **1731** with creation of a G-buffer at block **1733** by drawing all opaque (non-transparent) geometry and recording for each pixel its attributes that are necessary for lighting/shading computations. These involve positions, normal, and texture coordinates, other user-defined attributes, and/or the like.

[0188] To accelerate the lighting computations, at block **1735**, many applications compute lighting acceleration data that is commonly in the form of per-tile light lists that indicate, for each tile of pixels, which light sources are capable of contributing to the shading of those pixels. This is denoted as "light lists", but that other formats are also contemplated and the embodiments are not limited as such.

[0189] At block **1737**, a main lighting pass is performed, where this main lighting pass includes a fullscreen rendering pass that consumes the G-buffer data and light lists to compute the shading/lighting at each pixel. In some cases, this pass is divided into multiple sub-passes, for example, for each shader and/or light source or class of light source types. Applications that require transparent geometry typically draw such elements on top of the shaded image using blending. Finally, at block **1739**, applications typically involve one or more additional fullscreen passes that compute other, unrelated effects, such as fog, defocus blur, etc., and method **1730** ends at block **1741**.

[0190] FIG. **17**C illustrates a conventional transaction sequence **1750** for deferred shading. For brevity, many of the details previously discussed with reference to FIGS. **1-17**B may not be discussed or repeated hereafter. As illustrated, transaction sequence **1750** starts at block **1751** with drawing of all opaque (non-transparent) geometry and recording for each pixel, its attributes that are necessary for lighting/shading computations as part of a rasterization pass at block **1753**, resulting in G-buffer **1763**. At block **1755**, a fullscreen pass is performed including pre-pixel shading of G-buffer, which is then followed by another process of the rasteriztion pass including drawing of alpha at block **1757**. At block **1759**, as part of the fullscreen pass, post processing is performed which then results in displaying of content, such as images, using one or more display screens or devices at block **1761**. It is further illustrated that light lists **1765** are inputted into the processes of per-pixel shading G-buffer of block **1755** and drawing of alpha at block **1757**.

[0191] FIG. **18**A illustrates a method **1800** for multi-resolution deferred shading according to one embodiment. For brevity, many of the details previously discussed with reference to FIGS. **1-17**C may not be discussed or repeated hereafter. Method **1800** may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof, as facilitated by shading mechanism **1510** of FIG. **15**. The processes of method **1800** are illustrated in linear sequences for brevity and clarity in presentation; however, it is con-

templated that any number of them can be performed in parallel, asynchronously, or in different orders.

[0192] Method **1800** begins at block **1801** with rendering of a G-buffer at block **1803** and computing of per-tile light lists at block **1805**. As previously discussed with reference to FIG. **16**, in one embodiment, shading rate may be determined in a separate fullscreen pass prior to main light pass as shown in reference to blocks **1807** and **1809** and further illustrated in detail with reference to FIGS. **18C**-**18D**. Stated differently, in one embodiment, deferred lighting computations are moved into a TS, where the TS operates on a fullscreen PT with multiple mipmap levels, such as the base resolution of the PT may be the same as screen dimensions, while any subsequent mip levels may be relatively small. Further, in one embodiment, a fullscreen deferred shading pass (screen space), such as block **1737** of FIG. **17B**, is split into two shaders: 1) a pixel shader, as facilitated by computation logic **1607** of FIG. **16**, for determining a shading rate at block **1807** and triggering texel shading; and 2) a texel shader, as facilitated by shading pass logic **1609** of FIG. **16**, to perform the actual lighting or shading computations to facilitate a fullscreen lighting or shading pass (texture space) at block **1809**.

[0193] As previously discussed with reference to FIG. **16**, the shading rate may be controlled by selecting the mip level at which the texel shader executes, where this selection can be done at fine granularity, such as per pixel, allowing an application to shade finely near important features and more coarsely in other areas of the screen, where shading coarsely (at higher mip) may mean that each texel covers a larger screen space area. Method **1800** then continues with a fullscreen post-processing pass at block **1811** and ends at block **1813**.

[0194] FIG. **18B** illustrates a transaction sequence **1820** for multi-resolution deferred shading using a texel shader according to one embodiment. For brevity, many of the details previously discussed with reference to FIGS. **1**-**18A** may not be discussed or repeated hereafter. Transaction sequence **1820** may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof, as facilitated by shading mechanism **1510** of FIG. **15**. The processes of transaction sequence **1820** are illustrated in linear sequences for brevity and clarity in presentation; however, it is contemplated that any number of them can be performed in parallel, asynchronously, or in different orders.

[0195] Transaction sequence **1820** starts at block **1821** with drawing of all opaque (non-transparent) geometry and recording for each pixel, its attributes that are necessary for lighting/shading computations as part of a rasterization pass at block **1823**, resulting in G-buffer **1839**. As illustrated, in one embodiment, this G-buffer **1839** may then be used as an input for computing per-pixel shading rate at block **1825** and per-texel shading at block **1831**.

[0196] As aforementioned with reference to FIG. **16**, transaction sequence **1820** assumes a 2-pass texel shader technique, where texel shading is first triggered at block **1827** and then sampled in a separate fullscreen pass, such as fetch texel shading at block **1829**. In this embodiment, the shading rate is computed in a separate pass at block **1825** before the fullscreen shading pass (texel space) **1845** involving triggering of texel shading at block **1827**, fetching of texel shading at block **1829**, and per-texel shading of G-buf-

fer at block **1831**, resulting in writing of shaded texels to one or more PTs, such as PT **1841**, which may also be used as an input to fetch shading at block **1829**. In another embodiment, the shading rate may be computed as part of fullscreen shading pass **1845**, such as immediately before triggering of the texel shader at block **1827**.

[0197] Transaction sequence **1820** continues at block **1833** with the usual handling of transparent surfaces by blending transparent (e.g., alpha) objects on top of a final frame buffer after deferred shading, using light lists **1843** as an input, where light lists **1843** may also be used an input for per-texel shading of G-buffer at block **1831**. As illustrated, transaction sequence **1820** continues with post processing at block **1835** and ends with displaying of contents using one or more display devices at block **1837**.

[0198] FIG. **18C** illustrates a method **1850** for computing per-pixel shading rate according to one embodiment. For brevity, many of the details previously discussed with reference to FIGS. **1**-**18B** may not be discussed or repeated hereafter. Method **1850** may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof, as facilitated by shading mechanism **1510** of FIG. **15**. The processes of method **1850** are illustrated in linear sequences for brevity and clarity in presentation; however, it is contemplated that any number of them can be performed in parallel, asynchronously, or in different orders.

[0199] As an initial matter, it is contemplated and to be noted that in one embodiment, method **1850** reflects or represents block **1807** of FIG. **18A** and/or block **1825** of FIG. **18B**. Method **1850** begins at block **1851** with loading of external input (e.g., eye-tracking data, etc.) at block **1853**. Now, at block **1855**, a computational process is triggered for each pixel (x,y) on screen, which leads to loading of attributes at (x,y) from G-buffer at block **1857**. At block **1859**, shading rate (lod) is computed based on the attributes of block **1857** and/or any external inputs of block **1853**. At block **1861**, the shading rate (lod) is stored at pixel (x,y). At block **1863**, a determination is made as to whether there are any more pixels for which the computation need be performed. If yes, method **1850** continues at block **1855**. If not, method **1850** ends at block **1865**.

[0200] FIG. **18D** illustrates a method **1870** for performing fullscreen shading pass (texture space) according to one embodiment. For brevity, many of the details previously discussed with reference to FIGS. **1**-**18C** may not be discussed or repeated hereafter. Method **1870** may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a combination thereof, as facilitated by shading mechanism **1510** of FIG. **15**. The processes of method **1850** are illustrated in linear sequences for brevity and clarity in presentation; however, it is contemplated that any number of them can be performed in parallel, asynchronously, or in different orders.

[0201] As an initial matter, it is contemplated and to be noted that in one embodiment, method **1850** reflects or represents block **1809** of FIG. **18A** and/or block **1845** of FIG. **18B**. In the illustrated embodiment, method **1870** is shown as being divided into two following two methods or processes: trigger texel shading **1871** (corresponding to

block **1827** of FIG. **18**B) and fetch texel shading **1893** (corresponding to block **1829** of FIG. **18**B).

[0202] Method **1870** starts at block **1873** with determination of each pixel (x,y) on screen at block **1875**, followed by loading of shading rate (lod) (such as computed through an earlier process as set forth with respect to FIG. **18**A) at block **1877**. At block **1879**, filter footprint is then computed based on (x,y,lod) and subsequently used for each texel (u,v) in filter footprint at block **1881**. At block **1883**, a determination is made as to whether texel (u,v) is shaded. If not, texel is shaded (using G-buffer attributes, light lists, etc.) at block **1885**, while this shading is then stored to PT at block **1887**. Referring back to block **1883**, if texel (u,v) is shaded, method **1870** continues at block **1889** with another determination as to whether there are any more texels. If yes, method **1870** continues the process at block **1881**. If not, at block **1891**, yet another determination is made as to whether there are more pixels. If yes, method **1870** continues with the process of block **1875**. If not, the phase or portion method **1870** relating trigger texel shading **1871** moves into another phase or portion relating to fetch text shading **1893**.

[0203] In one embodiment, in fetch text shading **1893**, upon determining there are no more pixels as determined from block **1891**, a process is triggered for each pixel (x,y) on screen at block **1894** such that shading rate (lod) is loaded at block **1895**. At block **1896**, texel shading at (x,y,lod) is sampled to obtain filtered color, where the color is stored to pixel (x,y) in a frame buffer at block **1897**. At block **1898**, another determination is made as to whether there are more pixels. If yes, method **1870** continues with the process of block **1894**. If not, method **1870** ends at block **1899**.

[0204] References to "one embodiment", "an embodiment", "example embodiment", "various embodiments", etc., indicate that the embodiment(s) so described may include particular features, structures, or characteristics, but not every embodiment necessarily includes the particular features, structures, or characteristics. Further, some embodiments may have some, all, or none of the features described for other embodiments.

[0205] In the foregoing specification, embodiments have been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of embodiments as set forth in the appended claims. The Specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

[0206] In the following description and claims, the term "coupled" along with its derivatives, may be used. "Coupled" is used to indicate that two or more elements co-operate or interact with each other, but they may or may not have intervening physical or electrical components between them.

[0207] As used in the claims, unless otherwise specified the use of the ordinal adjectives "first", "second", "third", etc., to describe a common element, merely indicate that different instances of like elements are being referred to, and are not intended to imply that the elements so described must be in a given sequence, either temporally, spatially, in ranking, or in any other manner.

[0208] The following clauses and/or examples pertain to further embodiments or examples. Specifics in the examples may be used anywhere in one or more embodiments. The various features of the different embodiments or examples may be variously combined with some features included and others excluded to suit a variety of different applications. Examples may include subject matter such as a method, means for performing acts of the method, at least one machine-readable medium including instructions that, when performed by a machine cause the machine to performs acts of the method, or of an apparatus or system for facilitating hybrid communication according to embodiments and examples described herein.

[0209] Some embodiments pertain to Example 1 that includes an apparatus to facilitate multi-resolution deferred shading using texel shaders in computing environments, the apparatus comprising: pixel shading computation logic to facilitate computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and shading pass logic to facilitate texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

[0210] Example 2 includes the subject matter of Example 1, further comprising: division/splitting logic to split a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

[0211] Example 3 includes the subject matter of Example 1, wherein the pixel shading computation logic is further to facilitate a pixel shader to compute the shading rate in the first pass immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

[0212] Example 4 includes the subject matter of Example 1, wherein the shading pass logic is further to perform the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

[0213] Example 5 includes the subject matter of Example 1, wherein the texel shading operations further comprise fetch texel shading.

[0214] Example 6 includes the subject matter of Example 4, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

[0215] Example 7 includes the subject matter of Example 1, further comprising: detection/reception logic to detect one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and execution/forwarding logic to execute the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of contents having one or more images to be display using one or more display devices coupled to the apparatus.

[0216] Example 8 includes the subject matter of Example 3, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

[0217] Some embodiments pertain to Example 9 that includes a method for facilitating multi-resolution deferred

shading using texel shaders in computing environments, the method comprising: facilitating computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and facilitating texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

[0218] Example 10 includes the subject matter of Example 9, further comprising splitting a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

[0219] Example 11 includes the subject matter of Example 9, further comprising facilitating a pixel shader to compute the shading rate in the first pass immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

[0220] Example 12 includes the subject matter of Example 9, further comprising performing the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

[0221] Example 13 includes the subject matter of Example 9, wherein the texel shading operations further comprise fetch texel shading.

[0222] Example 14 includes the subject matter of Example 12, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

[0223] Example 15 includes the subject matter of Example 9, further comprising: detecting one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and executing the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of contents having one or more images to be display using one or more display devices coupled to the apparatus.

[0224] Example 16 includes the subject matter of Example 11, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

[0225] Some embodiments pertain to Example 17 includes a system comprising a computing device including a storage device having instructions, and a processor to execute the instructions to perform one or more operations comprising: facilitating computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and facilitating texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

[0226] Example 18 includes the subject matter of Example 17, wherein the one or more operations comprise splitting a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

[0227] Example 19 includes the subject matter of Example 17, wherein the one or more operations comprise facilitating a pixel shader to compute the shading rate in the first pass

immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

[0228] Example 20 includes the subject matter of Example 17, wherein the one or more operations comprise performing the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

[0229] Example 21 includes the subject matter of Example 17, wherein the texel shading operations further comprise fetch texel shading.

[0230] Example 22 includes the subject matter of Example 21, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

[0231] Example 23 includes the subject matter of Example 17, wherein the one or more operations comprise: detecting one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and executing the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of contents having one or more images to be display using one or more display devices coupled to the apparatus.

[0232] Example 24 includes the subject matter of Example 20, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

[0233] Some embodiments pertain to Example 25 includes an apparatus comprising: means for facilitating computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and means for facilitating texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

[0234] Example 26 includes the subject matter of Example 25, further comprising means for splitting a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

[0235] Example 27 includes the subject matter of Example 25, further comprising means for facilitating a pixel shader to compute the shading rate in the first pass immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

[0236] Example 28 includes the subject matter of Example 25, further comprising means for performing the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

[0237] Example 29 includes the subject matter of Example 25, wherein the texel shading operations further comprise fetch texel shading.

[0238] Example 30 includes the subject matter of Example 28, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

[0239] Example 31 includes the subject matter of Example 25, further comprising: means for detecting one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and means for executing the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of contents having one or more images to be display using one or more display devices coupled to the apparatus.

[0240] Example 32 includes the subject matter of Example 27, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

[0241] Example 33 includes at least one non-transitory or tangible machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims or examples 9-16.

[0242] Example 34 includes at least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims or examples 9-16.

[0243] Example 35 includes a system comprising a mechanism to implement or perform a method as claimed in any of claims or examples 9-16.

[0244] Example 36 includes an apparatus comprising means for performing a method as claimed in any of claims or examples 9-16.

[0245] Example 37 includes a computing device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

[0246] Example 38 includes a communications device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

[0247] Example 39 includes at least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[0248] Example 40 includes at least one non-transitory or tangible machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[0249] Example 41 includes a system comprising a mechanism to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[0250] Example 42 includes an apparatus comprising means to perform a method as claimed in any preceding claims.

[0251] Example 43 includes a computing device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[0252] Example 44 includes a communications device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[0253] The drawings and the forgoing description give examples of embodiments. Those skilled in the art will appreciate that one or more of the described elements may well be combined into a single functional element. Alterna- tively, certain elements may be split into multiple functional elements. Elements from one embodiment may be added to another embodiment. For example, orders of processes described herein may be changed and are not limited to the manner described herein. Moreover, the actions of any flow diagram need not be implemented in the order shown; nor do all of the acts necessarily need to be performed. Also, those acts that are not dependent on other acts may be performed in parallel with the other acts. The scope of embodiments is by no means limited by these specific examples. Numerous variations, whether explicitly given in the specification or not, such as differences in structure, dimension, and use of material, are possible. The scope of embodiments is at least as broad as given by the following claims.

What is claimed is:

1. An apparatus comprising:

pixel shading computation logic to facilitate computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and

shading pass logic to facilitate texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

2. The apparatus of claim 1, further comprising: division/ splitting logic to split a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

3. The apparatus of claim 1, wherein the pixel shading computation logic is further to facilitate a pixel shader to compute the shading rate in the first pass immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

4. The apparatus of claim 1, wherein the shading pass logic is further to perform the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

5. The apparatus of claim 1, wherein the texel shading operations further comprise fetch texel shading.

6. The apparatus of claim 4, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

7. The apparatus of claim 1, further comprising:

detection/reception logic to detect one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and

execution/forwarding logic to execute the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of con- tents having one or more images to be display using one or more display devices coupled to the apparatus.

8. The apparatus of claim 3, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

22

9. A method comprising:
facilitating computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and
facilitating texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

10. The method of claim 9, further comprising splitting a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

11. The method of claim 9, further comprising facilitating a pixel shader to compute the shading rate in the first pass immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

12. The method of claim 9, further comprising performing the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

13. The method of claim 9, wherein the texel shading operations further comprise fetch texel shading.

14. The method of claim 12, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

15. The method of claim 9, further comprising:
detecting one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and
executing the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of contents having one or more images to be display using one or more display devices coupled to the apparatus.

16. The method of claim 11, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

17. At least one machine-readable storage medium comprising a plurality of instructions, executed on a computing device, to facilitate the computing device to:

facilitate computation of shading rate in a first pass in a graphics pipeline, wherein the shading rate relates to a plurality of pixels; and
facilitate texel shading operations in a second pass using the shading rate, wherein the first pass is performed separate from and prior to the second pass.

18. The machine-readable medium of claim 17, wherein the computing device is further to split a fullscreen shading pass into the first pass and the second pass, wherein the fullscreen shading pass is performed in texture space.

19. The machine-readable medium of claim 17, wherein the computing device is further to facilitate a pixel shader to compute the shading rate in the first pass immediately before the second pass, wherein the shading rate is computed on per-pixel basis for each pixel of plurality of pixels or per-tile bases for groups of pixels of the plurality of pixels.

20. The machine-readable medium of claim 17, wherein the computing device is further to perform the texel shading operations on a procedural texture, wherein the texel shading operations comprise trigger a texel shader to facilitate per-texel shading of a geometry buffer.

21. The machine-readable medium of claim 17, wherein the texel shading operations further comprise fetch texel shading.

22. The machine-readable medium of claim 20, wherein the texel shader operations to query the procedural texture to determine whether relevant texels of a plurality of texels have already been shaded prior to trigger the texel shader.

23. The machine-readable medium of claim 17, herein the computing device is further to:
detect one or more stages of the graphics pipeline, wherein the one or more stages include rendering of the geometry buffer, computing per-tile light lists, the first phase, the second phase, and fullscreen post processing pass; and
execute the computation of the shading rate and the performance of the fullscreen shading pass to perform graphics processing of contents having one or more images to be display using one or more display devices coupled to the apparatus.

24. The machine-readable medium of claim 19, wherein the computation of the shading rate in the first pass further comprises loading attributes from the geometry buffer and analyzing the loaded attributes, and wherein the computation of the shading rate further comprises loading data from external sensors to facilitate shading rate computations, wherein the external sensors include an eye tracker.

* * * * *