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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 October 2015 (08.10.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/153170 A1

(51) International Patent Classification:
G09G 5/00 (2006.01)

(21) International Application Number:
PCT/US2015/021987

(22) International Filing Date:
23 March 2015 (23.03.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/246,062 5 April 2014 (05.04.2014) US

(71) Applicant: **SONY COMPUTER ENTERTAINMENT AMERICA LLC** [US/US]; 2207 Bridgepointe Parkway, San Mateo, California 94404 (US).

(72) Inventor: **CERNY, Mark Evan**; C/o Sony Computer Entertainment America LLC, 2207 Bridgepointe Parkway, San Mateo, California 94404 (US).

(74) Agent: **ISENBERG, Joshua D.**; JDI Patent, 809 Corporate Way, Fremont, California 94539 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Published:

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

(54) Title: GRADIENT ADJUSTMENT FOR TEXTURE MAPPING FOR MULTIPLE RENDER TARGETS WITH RESOLUTION THAT VARIES BY SCREEN LOCATION

(57) Abstract: In a computer graphics processing unit (GPU) having a shader and a texture unit the pixel shader is configured to receive or generate one or more sets of texture coordinates per pixel sample location. The pixel shader and texture unit between them are configured to calculate texture space gradient values for one or more primitives and generate and apply per-pixel gradient scale factors configured to modify the gradient values to smoothly transition them between regions of a display device having different pixel resolutions.

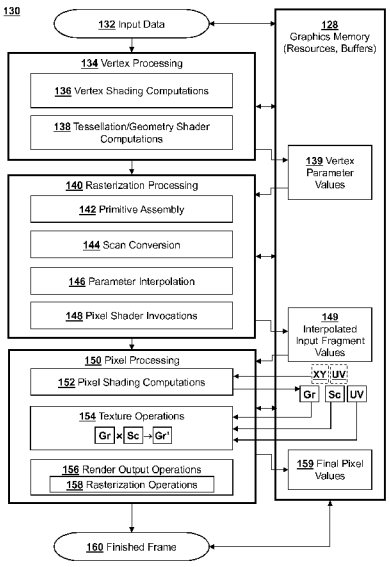


FIG. 1B

WO 2015/153170 A1

GRADIENT ADJUSTMENT FOR TEXTURE MAPPING FOR MULTIPLE RENDER TARGETS WITH RESOLUTION THAT VARIES BY SCREEN LOCATION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to commonly-assigned, co-pending US patent application number 14/246,064, to Tobias Berghoff, entitled "METHOD FOR EFFICIENT CONSTRUCTION OF HIGH RESOLUTION DISPLAY BUFFERS", (Attorney Docket No. SCEA13055US00), filed the same day as the present application, the entire contents of which are herein incorporated by reference.

This application is related to commonly-assigned, co-pending US patent application number 14/246,067, to Tobias Berghoff, entitled "GRAPHICS PROCESSING ENHANCEMENT BY TRACKING OBJECT AND/OR PRIMITIVE IDENTIFIERS", (Attorney Docket No. SCEA13056US00), filed April 5, 2014, the entire contents of which are herein incorporated by reference.

This application is related to commonly-assigned, co-pending US patent application number 14/246,068, to Mark Evan Cerny, entitled "GRADIENT ADJUSTMENT FOR TEXTURE MAPPING TO NON-ORTHONORMAL GRID", (Attorney Docket No. SCEA13057US00), filed April 5, 2014, the entire contents of which are herein incorporated by reference.

This application is related to commonly-assigned, co-pending US patent application number 14/246,061, to Tobias Berghoff, entitled "VARYING EFFECTIVE RESOLUTION BY SCREEN LOCATION BY CHANGING ACTIVE COLOR SAMPLE COUNT WITHIN MULTIPLE RENDER TARGETS", (Attorney Docket No. SCEA13058US00), filed April 5, 2014, the entire contents of which are herein incorporated by reference.

This application is related to commonly-assigned, co-pending US patent application number 14/246,063, to Mark Evan Cerny, entitled "VARYING EFFECTIVE RESOLUTION BY SCREEN LOCATION BY ALTERING RASTERIZATION PARAMETERS", (Attorney Docket No. SCEA13059US00), filed April 5, 2014, the entire contents of which are herein incorporated by reference.

This application is related to commonly-assigned, co-pending US patent application number 14/246,066, to Mark Evan Cerny, entitled "VARYING EFFECTIVE RESOLUTION BY SCREEN LOCATION IN GRAPHICS PROCESSING BY APPROXIMATING

PROJECTION OF VERTICES ONTO CURVED VIEWPORT", (Attorney Docket No. SCEA13060US00), filed April 5, 2014, the entire contents of which are herein incorporated by reference.

FIELD OF THE DISCLOSURE

Aspects of the present disclosure are related to computer graphics. In particular, the present disclosure is related to adjustment of gradients used for texture mapping.

BACKGROUND

Graphics processing typically involves coordination of two processors, a central processing unit (CPU) and a graphics processing unit (GPU). The GPU is a specialized electronic circuit designed to accelerate the creation of images in a frame buffer intended for output to a display. GPUs are used in embedded systems, mobile phones, personal computers, tablet computers, portable game devices, workstations, and game consoles. A GPU is typically designed to be efficient at manipulating computer graphics. GPU's often have a highly parallel processing architecture that makes the GPU more effective than a general-purpose CPU for algorithms where processing of large blocks of data is done in parallel.

The CPU may send the GPU instructions, commonly referred to as draw commands, that instruct the GPU to implement a particular graphics processing task, e.g. render a particular texture that has changed with respect to a previous frame in an image. These draw commands may be coordinated by the CPU with a graphics application programming interface (API) in order to issue graphics rendering commands that correspond to the state of the particular application's virtual environment.

In order to render textures for a particular program, a GPU may perform a series of processing tasks in a "graphics pipeline" to translate the visuals in the virtual environment into images that can be rendered onto a display. A typical graphics pipeline may include performing certain rendering or shading operations on virtual objects in the virtual space, transformation and rasterization of the virtual objects in the scene to produce pixel data suitable for output display, and additional rendering tasks on the pixels (or fragments) before outputting the rendered image on a display.

Virtual objects of an image are often described in virtual space in terms of shapes known as primitives, which together make the shapes of the objects in the virtual scene. For example, objects in a three-dimensional virtual world to be rendered may be reduced to a series of

distinct triangle primitives having vertices defined in terms of their coordinates in three-dimensional space, whereby these polygons make up the surfaces of the objects. Each polygon may have an associated index that can be used by the graphics processing system to distinguish a given polygon from other polygons. Likewise, each vertex may have an associated index that can be used to distinguish a given vertex from other vertices. A graphics pipeline may perform certain operations on these primitives to produce visuals for the virtual scene and transform this data into a two-dimensional format suitable for reproduction by the pixels of the display. The term graphics primitive information (or simply "primitive information"), as used herein, is used to refer to data representative of a graphics primitive. Such data includes, but is not limited to, vertex information (e.g., data representing vertex positions or vertex indices) and polygon information, e.g., polygon indices and information that associates particular vertices with particular polygons.

A GPU may perform rendering tasks of the graphics pipeline by implementing programs commonly known as shaders. A typical graphics pipeline may include vertex shaders, which may manipulate certain properties of the primitives on a per-vertex basis, as well as pixel shaders (also known as "fragment shaders"), which operate downstream from the vertex shaders in the graphics pipeline and may manipulate certain values on a per-pixel basis before transmitting the pixel data to a display. The fragment shaders may manipulate values relevant to applying textures to primitives. The pipeline may also include other shaders at various stages in the pipeline, such as geometry shaders that use the output of the vertex shaders to generate a new set of primitives, as well as compute shaders (CS) which may implemented by a GPU to perform certain other general computational tasks.

Part of the process of mapping textures to primitives involves calculating gradients in texture space from pixel locations in screen space. The gradient calculation often assumes that the pixel locations are based on a square orthonormal grid.

It is within this context that the present disclosure arises.

BRIEF DESCRIPTION OF THE DRAWINGS

The teachings of the present disclosure can be readily understood by considering the following detailed description in conjunction with the accompanying drawings, in which:

FIG. 1A is a block diagram of a graphics processing system in accordance with aspects of the present disclosure.

FIG. 1B is a block diagram of a graphics processing pipeline.

FIGs. 2A-2D depict a portion of a display having regions of differing resolution accordance with an aspect of the present disclosure.

FIG. 2E illustrates a generalized example of per-pixel gradient scale correction in accordance with an aspect of the present disclosure.

DESCRIPTION OF THE DRAWINGS

Although the following detailed description contains many specific details for the purposes of illustration, anyone of ordinary skill in the art will appreciate that many variations and alterations to the following details are within the scope of the invention. Accordingly, the exemplary embodiments of the invention described below are set forth without any loss of generality to, and without imposing limitations upon, the claimed invention.

Introduction

In certain graphics applications bitmapped textures are "painted" onto the polygon. In such a case each pixel value drawn by the output device is determined from one or more pixels sampled from the texture. As used herein, a bitmap generally refers to a data file or structure representing a generally rectangular grid of pixels, or points of color, on a computer monitor, paper, or other display device. The color of each pixel is individually defined. For example, a colored pixel may be defined by three bytes—one byte each for red, green and blue. A bitmap typically corresponds bit for bit with data formats supported by device texture sampling units, which may typically include a range of options including various bit depths per channel or block compression, probably in the same format as it would be stored in the display's video memory or maybe as a device independent bitmap. A bitmap is characterized by the width and height of the image in pixels and the number of bits per pixel, which determines the number of colors it can represent.

The process of transferring a texture bitmap to a surface often involves the use of texture MIP maps (also known as mipmaps). The letters "MIP" in the name are an acronym of the Latin phrase *multum in parvo*, meaning "much in a small space". Such mipmaps are pre-calculated, optimized collections of bitmap images that accompany a main texture, intended to increase rendering speed and reduce aliasing artifacts.

Each bitmap image of the mipmap set is a version of the main texture, but at a certain reduced level of detail (LOD). Although the main texture would still be used when the view

is sufficient to render it in full detail, the graphics hardware rendering the final image switches to a suitable mipmap level (or interpolates between the two nearest levels) when the texture is viewed from a distance, or at a small size. Rendering speed increases since the number of texture pixels ("texels") being processed can be much lower and their distribution in memory more coherent than with simple textures. Artifacts may be reduced since the mipmap images are effectively already anti-aliased, taking some of the burden off the real-time rendering hardware.

The blending between mipmap levels typically involves some form of texture filtering. As used herein, texture filtering refers to a method used to map texels (pixels of a texture) to points on a 3D object. A simple texture filtering algorithm may take a point on an object and look up the closest texel to that position. The resulting point then gets its color from that one texel. This simple technique is sometimes referred to as *nearest neighbor* filtering. More sophisticated techniques combine more than one texel per point. The most often used algorithms in practice are bilinear filtering and trilinear filtering using mipmaps. Anisotropic filtering and higher-degree methods, such as quadratic or cubic filtering, result in even higher quality images.

Textures are typically square and have side lengths equal to a power of 2. If, e.g., a texture has a basic size of 256 by 256 pixels, then the associated mipmap set may contain a series of 8 images, each half the size of the previous one: 128×128 pixels, 64×64, 32×32, 16×16, 8×8, 4×4, 2×2, and 1×1 (a single pixel). If, for example, this texture is mapped onto a 40×40 pixel portion of a screen space, then an interpolation of the 64×64 and the 32×32 mipmaps would be used. As used herein the term “screen space” refers generally to the set of coordinates used by the display buffer in the graphics pipeline.

The key operation in the process of determining the appropriate mipmap level involves determining the area covered in texture coordinate space (sometimes referred to as UV coordinate space) for a corresponding area of pixel locations from screen space (sometimes referred to as XY coordinate space). In general terms, screen space gradients of interpolated or computed texture UV coordinates are calculated from U and V values sampled at XY space pixel locations in the relevant portion of the scene. In some implementations, a texture coordinate gradient is determined for each screen space direction X and Y by calculating the change in texture coordinates occurring when the screen X coordinate changes and the screen Y is fixed (sometimes referred to as du_dx, dv_dx) and the change in texture coordinates

occurring when the screen Y coordinate changes and the screen X is fixed (sometimes referred to as du_dy, dv_dy). This texture coordinate gradient calculation may optionally include corrections for non-orthonormality of the sample grid. For non-anisotropic texture lookups, the gradient with the larger magnitude among these two is used to select level of detail (LOD). For anistropic texturing, the smaller magnitude gradient is used to select the LOD, and the texture is sampled in a line corresponding to the larger magnitude gradient.

Also note that the above calculation can be generalized to 1, 2, 3, or more texture coordinate dimensions. Typical hardware calculates a 1D gradient in U-space or a 2D gradient in UV-space or a 3D gradient in UVW-space depending on the texture dimensionality. Thus, aspects of the present disclosure are not limited to implementations involving two texture coordinate dimensions.

The process of determining the appropriate MIP level of detail is, however, based on an assumption that the relevant portion of the virtual space onto which the texture to be applied is a regular arrangement of samples, i.e., the sample points within screen pixels are evenly spaced across the entirety of screen space in the vertical and horizontal directions. However, visual artifacts can arise from discontinuities in the sample pattern, for instance at a boundary between screen areas having different sample spacing in screen space. At such boundaries, locally correct texture filtering on either side of the boundary may access significantly different MIP levels of detail, producing a noticeable change in the appearance of the output image across the boundary. In such situations, the gradients must be adjusted on each side of the boundary in order to produce smoothly varying texture filtering and thereby reduce the visibility of the boundary.

System and Apparatus

Aspects of the present disclosure include graphics processing systems that are configured to implement gradient adjustment in texture mapping. By way of example, and not by way of limitation, FIG. 1A illustrates a block diagram of a computer system **100** that may be used to implement graphics processing according to aspects of the present disclosure. According to aspects of the present disclosure, the system **100** may be an embedded system, mobile phone, personal computer, tablet computer, portable game device, workstation, game console, and the like.

The system **100** generally may include a central processor unit (CPU) **102**, a graphics processor unit (GPU) **104**, and a memory **108** that is accessible to both the CPU and GPU. The CPU **102** and GPU **104** may each include one or more processor cores, e.g., a single core, two cores, four cores, eight cores, or more. The memory **108** may be in the form of an integrated circuit that provides addressable memory, e.g., RAM, DRAM, and the like. The memory **108** may include graphics memory **128** that may store graphics resources and temporarily store graphics buffers **105** of data for a graphics rendering pipeline. The graphics buffers **105** may include, e.g., vertex buffers for storing vertex parameter values, index buffers for holding vertex indices, depth buffers (e.g., Z-buffers) for storing depth values of graphics content, stencil buffers, frame buffers for storing completed frames to be sent to a display, and other buffers. In the example shown in FIG. 1A, the graphics memory **128** is shown as part of the main memory. In alternative implementations, the graphics memory could be a separate component, possibly integrated into the GPU **104**.

By way of example, and not by way of limitation, the CPU **102** and GPU **104** may access the memory **108** using a data bus **109**. In some cases, it may be useful for the system **100** to include two or more different buses. The memory **108** may contain data that can be accessed by the CPU **102** and GPU **104**. The GPU **104** may include a plurality of compute units configured to perform graphics processing tasks in parallel. Each compute unit may include its own dedicated local memory store, such as a local data share.

The CPU may be configured to execute CPU code $\mathbf{103}_C$, which may include an application that utilizes graphics, a compiler and a graphics API. The graphics API can be configured to issue draw commands to programs implemented by the GPU. The CPU code $\mathbf{103}_C$ may also implement physics simulations and other functions. The GPU **104** may be configured to operate as discussed above. In particular, the GPU may execute GPU code $\mathbf{103}_G$, which may implement shaders, such as compute shaders **CS,** vertex shaders **VS**, and pixel shaders **PS**, as discussed above. To facilitate passing of data between the compute shaders **CS** and the vertex shaders **VS** the system may include one or more buffers **105**, which may include a frame buffer **FB**. The GPU code $\mathbf{103}_G$ may also optionally implement other types of shaders (not shown), such as pixel shaders or geometry shaders. Each compute unit may include its own dedicated local memory store, such as a local data share. The GPU **104** may include one or more texture units **106** configured to perform certain operations for applying textures to primitives as part of a graphics pipeline.

According to aspects of the present disclosure a pixel shader **PS** and a texture unit **106** are configured to generate one or more texture coordinates **UV** and potentially also texture space gradient values for each coordinate **Gr**, for one or more corresponding pixel sample locations. These gradient values may potentially be corrected for non-orthonormality of the sample grid. While these gradient values **Gr** provide locally correct texture filtering, in some configurations, there may be abrupt transitions between screen areas having different pixel sample configurations which may become visible in the output image as an abrupt change in texture filtering appearance. According to aspects of the present disclosure a pixel shader **PS** may calculate and supply the texture unit **106** with screen space gradient scale factors **Sc** which the texture unit **106** may apply as a linear screen axis aligned scale transformation to the gradients **Gr** to obtain adjusted gradient values **Gr'**. These gradient scale factors **Sc** may be used to modify the gradient values **Gr** progressively leading up to a screen area boundary in order to smoothly transition them between regions of the display device **116** having different pixel resolutions.

By way of example, and not by way of limitation, the texture units **106** may be implemented as special purpose hardware, such as an application-specific integrated circuit (ASIC), Field Programmable Gate Array (FPGA), or a system on chip (SoC or SOC).

As used herein and as is generally understood by those skilled in the art, an application-specific integrated circuit (ASIC) is an integrated circuit customized for a particular use, rather than intended for general-purpose use.

As used herein and as is generally understood by those skilled in the art, a Field Programmable Gate Array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an ASIC.

As used herein and as is generally understood by those skilled in the art, a system on a chip or system on chip (SoC or SOC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analog, mixed-signal, and often radio-frequency functions—all on a single chip substrate. A typical application is in the area of embedded systems.

A typical SoC includes the following hardware components:

One or more processor cores (e.g., microcontroller, microprocessor or digital signal processor (DSP) cores.

Memory blocks, e.g., read only memory (ROM), random access memory (RAM), electrically erasable programmable read-only memory (EEPROM) and flash memory.

Timing sources, such as oscillators or phase-locked loops.

Peripherals, such as counter-timers, real-time timers, or power-on reset generators.

External interfaces, e.g., industry standards such as universal serial bus (USB), FireWire, Ethernet, universal asynchronous receiver/transmitter (USART), serial peripheral interface (SPI) bus.

Analog interfaces including analog to digital converters (ADCs) and digital to analog converters (DACs).

Voltage regulators and power management circuits.

These components are connected by either a proprietary or industry-standard bus. Direct Memory Access (DMA) controllers route data directly between external interfaces and memory, bypassing the processor core and thereby increasing the data throughput of the SoC.

A typical SoC includes both the hardware components described above, and executable instructions (e.g., software or firmware) that controls the processor core(s), peripherals and interfaces.

According to aspects of the present disclosure, some or all of the functions of the texture units **106** may alternatively be implemented by appropriately configured software instructions executed by a software programmable general purpose computer processor. Such instructions may be embodied in a computer-readable medium, e.g., memory **108** or storage device **115**.

The system **100** may also include well-known support functions **110**, which may communicate with other components of the system, e.g., via the bus **109**. Such support functions may include, but are not limited to, input/output (I/O) elements **111**, power supplies (P/S) **112**, a clock (CLK) **113** and cache **114**. In addition to the cache **114**, the GPU **104** may include its own GPU cache $\mathbf{114_G}$, and the GPU may be configured so that programs running on the GPU **104** can read-through or write-though the GPU cache $\mathbf{114_G}$.

The system **100** may optionally include a mass storage device **115** such as a disk drive, CD-ROM drive, flash memory, tape drive, or the like to store programs and/or data. The system **100** may also optionally include a display device **116** to present rendered graphics **117** to a user and user interface unit **118** to facilitate interaction between the system **100** and a user. The display device **116** may be in the form of a flat panel display, head mounted display (HMD), cathode ray tube (CRT) screen, projector, or other device that can display visible text, numerals, graphical symbols or images. The display **116** may display rendered graphic images **117** processed in accordance with various techniques described herein. The user interface **118** may include a keyboard, mouse, joystick, light pen, game controller, or other device that may be used in conjunction with a graphical user interface (GUI). The system **100** may also include a network interface **120** to enable the device to communicate with other devices over a network **122**. The network **122** may be, e.g., a local area network (LAN), a wide area network such as the internet, a personal area network, such as a Bluetooth network or other type of network. These components may be implemented in hardware, software, or firmware, or some combination of two or more of these.

Graphics Pipeline

According to aspects of the present disclosure, the system **100** is configured to implement portions of a graphics rendering pipeline. FIG. 1B illustrates an example of a graphics rendering pipeline **130** in accordance with aspects of the present disclosure.

The rendering pipeline **130** may be configured to render graphics as images that depict a scene having a two-dimensional, or preferably three-dimensional geometry in virtual space (sometime referred to herein as "world space"). The early stages of the pipeline may include operations performed in virtual space before the scene is rasterized and converted to screen space as a set of discrete picture elements suitable for output on the display device **116**. Throughout the pipeline, various resources contained in the graphics memory **128** may be utilized at the pipeline stages and inputs and outputs to the stages may be temporarily stored in buffers contained in the graphics memory before the final values of the images are determined.

The rendering pipeline may operate on input data **132**, which may include one or more virtual objects defined by a set of vertices that are set up in virtual space and have geometry that is defined with respect to coordinates in the scene. The early stages of the pipeline may include what is broadly categorized as a vertex processing stage **134** in FIG. 1B, and this may include

various computations to process the vertices of the objects in virtual space. This may include vertex shading computations **136**, which may manipulate various parameter values of the vertices in the scene, such as position values (e.g., X-Y coordinate and Z-depth values), color values, lighting values, texture coordinates, and the like. Preferably, the vertex shading computations **136** are performed by one or more programmable vertex shaders. The vertex processing stage may optionally include additional vertex processing computations, such as tessellation and geometry shader computations **138** which may be optionally used to generate new vertices and new geometries in virtual space. Once the stage referred to as vertex processing **134** is complete, at this stage in the pipeline the scene is defined by a set of vertices which each have a set of vertex parameter values **139**.

The pipeline **130** may then proceed to rasterization processing stages **140** associated with converting the scene geometry into screen space and a set of discrete picture elements, i.e., pixels. The virtual space geometry may be transformed to screen space geometry through operations that may essentially compute the projection of the objects and vertices from virtual space to the viewing window (or "viewport) of the scene. The vertices may define a set of primitives.

The rasterization processing stage **140** depicted in FIG. 1B may include primitive assembly operations **142**, which may set up the primitives defined by each set of vertices in the scene. Each vertex may be defined by an index, and each primitive may be defined with respect to these vertex indices, which may be stored in index buffers in the graphics memory **128**. The primitives may preferably include at least triangles defined by three vertices each, but may also include point primitives line primitives, and other polygonal shapes. During the primitive assembly stage **142**, certain primitives may optionally be culled. For example, those primitives whose indices indicate a certain winding order may be considered to be back-facing and may be culled from the scene.

By way of example, and not by way of limitation, where the primitives are in the form of triangles defined by vertices in three dimensional virtual space, the primitive assembly determines where on the screen of the display **116** each triangle is located. Clipping and screen space transformation operations are typically performed by the primitive assembly unit **142**.

After primitives are assembled, the rasterization processing stages may include scan conversion operations **144**, which may sample the primitives at each pixel and generate fragments (sometimes referred to as pixels) from the primitives for further processing when the samples are covered by the primitive. The scan conversion operations include operations that take a primitive that has been converted to screen space coordinates and determines which pixels are part of that primitive. In some implementations, multiple samples for each pixel are taken within the primitives during the scan conversion operations **144**, which may be used for anti-aliasing purposes. In certain implementations, different pixels may be sampled differently. For example, some edge pixels may contain a lower sampling density than center pixels to optimize certain aspects of the rendering for certain types of display device **116**, such as head mounted displays (HMDs). The fragments (or "pixels") generated from the primitives during scan conversion **144** may have parameter values that may be interpolated to the locations of the pixels from the vertex parameter values **139** of the vertices of the primitive that created them. The rasterization stage **140** may include parameter interpolation operations **146** stage to compute these interpolated fragment parameter values **149**, which may be used as inputs for further processing at the later stages of the pipeline.

The pipeline **130** may include further pixel processing operations, indicated generally at **150** in FIG. 1B, to further manipulate the interpolated parameter values **149** and perform further operations determining how the fragments contribute to the final pixel values for display. Some of these pixel processing tasks may include pixel shading computations **152** that may be used to further manipulate the interpolated parameter values **149** of the fragments. The pixel shading computations may be performed by a programmable pixel shader, and pixel shader invocations **148** may be initiated based on the sampling of the primitives during the rasterization processing stages **140**. The pixel shading computations **152** may output values to one or more buffers **105** in graphics memory **128**, sometimes referred to as render targets, or if multiple, as multiple render targets (MRTs).

MRTs allow pixel shaders to optionally output to more than one render target, each with the same screen dimensions but potentially with a different pixel format. Render target format limitations often mean that any one render target can only accept up to four independent output values (channels) and that the formats of those four channels are tightly tied to each other. MRTs allow a single pixel shader to output many more values in a mix of different formats. The formats of render targets are "texture-like", in that they store values per screen space pixel, but, for various performance reasons, render target formats are becoming more

specialized in recent hardware generations, sometimes (but not always) requiring what is called a "resolve" to reformat the data before it is compatible with being read in by the texture units **106**.

The pixel processing **150** may generally culminate in render output operations **156**, which may include what are commonly known as raster operations (ROP). Rasterization Operations (ROP) is simply run multiple times per pixel, once for each render target among the multiple render targets (MRTs). During the output operations **156**, the final pixel values **159** may be determined in a frame buffer, which may optionally include merging fragments, applying stencils, depth tests, and certain per sample processing tasks. The final pixel values **159** include the collected output to all active render targets (MRTs). The GPU **104** uses the final pixel values **159** to make up a finished frame **160**, which may optionally be displayed on the pixels of the display device **116** in real-time.

The output operations **150** may also include texture mapping operations **154**, which may be performed to some extent by one or more pixel shaders **PS** and to some extent by the texture units **106**. The pixel shader computations **152** include calculating texture coordinates **UV** from screen space coordinates **XY**, and sending the texture coordinates to the Texture Operations **154**, and receiving texture data **TX**. The texture coordinates **UV** could be calculated from the screen space coordinates **XY** in an arbitrary fashion, but typically are calculated from interpolated input values or sometimes from the results of previous texture operations. Gradients **Gr** are often directly calculated from quads of texture coordinates by the texture units **106** (Texture Operations hardware units), but can optionally be calculated explicitly by the pixel shader computations **152** and passed to the texture operations **154** rather than relying on the texture units **106** to perform the default calculation.

The texture operations **154** generally include the following stages, which can be performed by some combination of a pixel shader **PS** and a texture unit **106**. First, one or more texture coordinates **UV** per pixel location **XY** are generated and used to provide a coordinate set for each texture mapping operation. Then, texture space gradient values **Gr** for pixel sample locations are generated, potentially including corrections for non-orthonormality of the sample grid. Finally, the gradients **Gr** are modified by per-pixel screen space gradient scale factors **Sc** supplied by the pixel shader **PS** in order to produce the final texture space gradients **Gr'** used for texture filtering operations. The gradient scale factors **Sc** may be chosen to produce gradient values **Gr'** which smoothly transition between regions of the display device **116** having different pixel resolutions or sample distributions.

In some implementations, the pixel shader **PS** can generate the texture coordinates **UV** and gradient scale factors **Sc** per pixel location **XY** and provide a coordinate set for each texture mapping operation to the texture unit **106**, which may generate texture space gradient values **Gr** and modify them to produce corrected texture space gradient values **Gr'**.

In other implementations, the pixel shader **PS** could calculate the texture space coordinates **UV**, explicit differences **Gr** from the pixel locations **XY**, and gradient scale factors **Sc** and pass all of these values to the texture unit **606** and indicate to the texture unit **606** that it must still perform any corrections for non-orthonormality normally, then apply the gradient scale factors **Sc** to get the adjusted gradient values **Gr'**.

In other alternative implementations, the pixel shader **PS** could calculate the texture space coordinates **UV** and explicit corrected gradients **Gr'** and pass those to the texture unit, indicating to the texture unit **106** that any required corrections have already been applied in software and that the corrected gradients **Gr'** should be used as is to select the LOD.

Per-Pixel Gradient Adjustment

Aspects of the present disclosure are directed to adjustment of gradients **Gr** used by the texture unit **106** to determine the mipmap level (LOD) for a texture to be applied to a primitive in the graphics pipeline. The basic concept is illustrated in FIG. 2E. FIG. 2E depicts the texture UV coordinates of four pixel samples in two orthogonal quads (**E0** and **E1**). In quad **E0**, all samples points in XY space lie on an orthonormal grid with spacing of one screen pixel horizontally and vertically.

In quad **E0**, texture gradients **Gr** in UV space are trivially computed from the texture coordinates **UV**. The texture gradients **Gr** may be expressed mathematically in terms of the texture coordinates **UV** of the upper left, upper right and lower left pixels, respectively (u0, v0), (u1, v1) and (u2, v2), as follows:

$$du_dx = u1-u0$$

$$dv_dx = v1-v0$$

$$du_dy = u2-u0$$

$$dv_dy = v2-v0$$

These calculations of du_dx=u1-u0, etc. can be performed by the texture unit hardware **106** if the pixel shader **PS** does not choose to override them with software calculated values. The

texture gradients **Gr** are then used by the texture unit **106** to determine the mipmap LOD level to sample.

According to aspects of the present disclosure the gradients **Gr** du_dx, dv_dx, du_dy, dv_dy are adjusted on a per-pixel basis to account for discontinuities in pixel resolution across a display. For example, in certain display configurations (e.g., head mounted display (HMD) applications), it is advantageous to use different resolutions for different parts of the display. In such cases it is very advantageous to scale the gradients on a per-pixel basis to smooth out the change in texture filtering at or near boundaries between adjacent regions of different resolution. The basic concept is illustrated in FIGs. 2A-2C.

As shown in FIG. 2A and FIG. 2B the display area may be divided into two or more regions **201**, **202**, **203**, **204**, **205**, **206**, **207**, **208**, and **209** of different pixel resolution. Each region may have a resolution that is related to the solid angle subtended by the region of the display, e.g., in the case of a head mounted display (HMD). By way of example, and not by way of limitation, the central region **205** may have a nominal or standard pixel resolution R_0. Edge regions **202**, **204**, **206**, **208** could have half the standard resolution $\frac{1}{2}R_0$, e.g., half the pixels in these regions would be "turned off" or not rendered on the display. Corner regions **201**, **203**, **207**, and **209** could have on quarter the standard resolution $\frac{1}{4}R_0$, e.g., three quarters of the pixels in these regions would be "turned off" or not rendered on the display. FIG. 2B shows these different regions drawn to different sizes according to the resolution of each region.

Gradients might need to be adjusted for pixels near the boundaries between adjacent regions. For example, in FIG. 2C the lines marking the boundaries between the sections have been removed. However, discontinuities in texture filtering between neighboring regions make the boundaries visible. The GPU can be configured to adjust gradients for pixels near the boundaries between regions to smooth out the discontinuities, making them less visible. For example, as shown in FIG. 2D, horizontal gradients for selected pixels **211**,**212** in the full resolution central region **205** in the row **D-D'** in FIG. 2C can be scaled so as to produce progressively blurrier texture filtering towards the half resolution edge regions **204**, **206**, such that two pixels immediately adjacent across the border would access approximately the same MIP LOD level.

Alternatively, gradients in the half resolution edge regions **204**,**206** may also be scaled so as to become progressively sharper towards the full resolution central region **205**.

Generalized adjustment of the gradients to account for differing resolution can be understood with reference to FIG. 2E, which depicts in UV space the texture coordinates for two adjacent quads of pixels **E0** and **E1** across a boundary in horizontal screen area, also indicated by the correspondingly labeled rectangles in screen space in FIG. 2C. In FIG. 2E, the halving of the horizontal density of samples between **E0** and **E1** produces (du_dx, dv_dx) for **E1** that have approximately twice the length of the (du_dx, dv_dx) calculated for **E0**. This may in turn result in selection of a lower level of detail for quad **E1** than for **E0**. It should be noted that these gradient calculations for quad **E0** and quad **E1** are locally correct, in that each would correctly select a mipmap LOD that would cover each screen pixel with approximately one texture pixel ("texel"), but that this abrupt change in texture filtering may produce a visible change in the appearance of the output image across the boundary between quads with sample patterns like **E0** and those with patterns like **E1**. The visibility of this boundary may be reduced by choosing to adjust gradient values in pixels near to the boundary in order to produce a smoothly varying selection of mipmap LOD.

Gradients in UV space (du_dx, dv_dx), (du_dy, dv_dy) are calculated normally for each quad, which might in some other cases include corrections for non-orthonormality of the local sample distribution. The gradients can then be multiplied by per-pixel gradient scale factors scaleX, scaleY on a per-pixel basis to produce the final adjusted gradients (du_dx', dv_dx'), (du_dy',dv_dy') that the texture unit uses to determine the appropriate mipmap level:

$$du_dx' = du_dx * scaleX$$
$$du_dy' = du_dy * scaleY$$
$$dv_dx' = dv_dx * scaleX$$
$$dv_dy' = dv_dy * scaleY$$

The gradient scale factors scaleX and scaleY, would be calculated by the pixel shader **PS** and passed to the texture unit **106**, which would use them to calculate the final adjusted gradients.

It should be understood that the above gradient scale correction is applied identically to each texture coordinate U and V, and so can be extended trivially to 1 dimensional coordinates (U), 2 dimensional coordinates (UV), or 3 dimensional coordinates (UVW).

Suitable values for gradient scale factors scaleX, scaleY can be determined empirically by iterating the gradient scale factor values applied to selected pixels proximate the boundaries of neighboring regions of different pixel resolution. A range of suitable values for the

gradient scale factors can be determined from the relative pixel resolutions of the neighboring regions. For example, along row **D-D'** the pixel resolution changes from ½R in region **204** to R in region **205** and again to ½R in region **206**. The gradient scale value scaleX would transition from roughly 1 to roughly 2 over several pixels in region **205** proximate the boundaries with regions **204** and **206**. In FIG. 2E, it can be seen that applying a gradient scaleX factor of 2 to quad **E0** from region **205** would result in (du_dx', dv_dx') which roughly match the gradients calculated for adjacent quad **E1** from region **206**, thus creating continuity in the scale corrected gradients and texture filtering LOD selection. Similarly, gradient scale values scaleX, scaleY would transition from a value of roughly 1 to a value of roughly 2 over several pixels in region **208** proximate the boundaries with regions **207** and **209**. The gradient scale factors may be varied between the two values over several pixels, e.g., roughly 4 to 8 pixels.

The texture unit **106** may use the final adjusted gradients to select the appropriate LOD to apply to one or more primitives from the final adjusted gradients.

Additional Aspects

An additional aspect of the present disclosure include a graphics processing method, comprising: receiving or generating one or more sets of texture coordinates per pixel sample location; calculating texture space gradient values for one or more primitives; and generating and applying per-pixel gradient scale factors configured to modify the gradient values to smoothly transition them between regions of a display device having different pixel resolutions.

Another additional aspect is a graphics processing system configured to implement the forgoing method.

Yet another additional aspect is a computer-readable medium having computer executable instructions embodied therein that, when executed, implement the foregoing method.

A further aspect is an electromagnetic or other signal carrying computer-readable instructions for performing the foregoing method.

Another further aspect is a computer program product downloadable from a communication network and/or stored on a computer-readable and/or microprocessor-executable medium, characterized in that it comprises program code instructions for implementing the foregoing method.

While the above is a complete description of the preferred embodiment of the present invention, it is possible to use various alternatives, modifications and equivalents. Therefore, the scope of the present invention should be determined not with reference to the above description but should, instead, be determined with reference to the appended claims, along with their full scope of equivalents. Any feature described herein, whether preferred or not, may be combined with any other feature described herein, whether preferred or not. In the claims that follow, the indefinite article "A", or "An" refers to a quantity of one or more of the item following the article, except where expressly stated otherwise. The appended claims are not to be interpreted as including means-plus-function limitations, unless such a limitation is explicitly recited in a given claim using the phrase "means for."

WHAT IS CLAIMED IS:

1. A computer graphics system, comprising:
a graphics processing unit (GPU) having a pixel shader and a texture unit;
wherein the pixel shader is configured to receive or generate one or more sets of texture coordinates per pixel sample location and wherein the pixel shader and texture unit between them are configured to calculate texture space gradient values for one or more primitives, and generate and apply per-pixel gradient scale factors configured to modify the gradient values to smoothly transition them between regions of a display device having different pixel resolutions.

2. The system of claim 1, wherein the texture space gradient values include corrections for non-orthonormality of a screen space sample distribution.

3. The system of claim 1, wherein the texture unit is an application specific integrated circuit (ASIC), field programmable gate array (FPGA), or system on chip (SOC).

4. The system of claim 1, wherein the texture unit is further configured to select a level of detail from a plurality of levels of detail for a texture to be applied to one or more primitives from the adjusted gradient values.

5. The system of claim 4, wherein the texture unit is further configured to apply the texture to the one or more primitives.

6. The system of claim 5, further comprising a display unit coupled to the GPU, wherein the display unit is configured to display images that include the texture applied to the one or more primitives.

7. The system of claim 1, wherein the texture unit calculates texture space gradient values from the texture coordinates supplied by the pixel shader and generates and applies the per-pixel gradient scale factors to obtain scale corrected gradient values.

8. The system of claim 1, wherein the pixel shader calculates the per-pixel gradient scale factors and provides them to the texture unit with the texture coordinates.

9. The system of claim 8, wherein the texture unit calculates texture space gradient values from the texture coordinates supplied by the pixel shader and then applies the supplied gradient scale factors to the texture space gradients to obtain adjusted gradient values.

10. The system of claim 1, wherein the pixel shader calculates the texture space gradients from the texture coordinates and passes them with the texture coordinates to the texture unit, wherein the pixel shader indicates to the texture unit that the texture unit must correct the texture gradient values for any non-orthonormality of the sample grid.

11. The system of claim 10 wherein the pixel shader also determines the per-pixel gradient scale factors and provides them to the texture unit, wherein the pixel shader notifies the texture unit that the texture unit must apply the gradient scale factors to the gradients to obtain the adjusted gradients.

12. The system of claim 1, wherein the pixel shader calculates the adjusted texture gradients from the texture coordinates and passes them along with the texture space coordinates to the texture unit, wherein the pixel shader indicates to the texture unit that the adjusted gradients should be used as is to select a level of detail for a texture to be applied to a primitive.

13. In a computer graphics processing unit (GPU) having a pixel shader and a texture unit, the texture unit being implemented in hardware, a graphics processing method, comprising:

the pixel shader generating one or more texture coordinates per pixel location to provide a coordinate set for one or more texture mapping operations;

between the pixel shader and texture unit calculating gradient values from the texture coordinates;

between the pixel shader and texture unit determining gradient scale factors configured to adjust corresponding gradient values to corresponding adjusted gradient values and

between the pixel shader and texture unit applying the gradient scale factors to the gradient values, wherein the adjustment factors are configured to modify the gradient values to smoothly transition them between regions of a display device having different pixel resolutions.

14. The method of claim 13 wherein the texture space gradient values include corrections for non-orthonormality of a screen space sample distribution

15. The method of claim 13, further comprising selecting a level of detail from a plurality of levels of detail for a texture to be applied to one or more primitives from the adjusted gradient values with the texture unit.

16. The method of claim 13, further comprising, applying the texture to the one or more primitives with the texture unit.

17. The method of claim 16, further comprising displaying images that include the texture applied to the one or more primitives with a display unit.

18. The method of claim 17, wherein the pixel shader generates the texture coordinates per pixel location and provides a coordinate set for each texture mapping operation to the texture unit.

19. The method of claim 18, wherein the texture unit calculates texture gradients from the texture coordinates supplied by the pixel shader, and generates and applies the gradient scale factors to obtain the corrected gradient values.

20. The method of claim 13, wherein the pixel shader calculates the gradient scale factors and provides them to the texture unit with the texture coordinates.

21. The method of claim 20, wherein the texture unit calculates texture space gradient values from the texture coordinates supplied by the pixel shader, and then applies the supplied gradient scale factors to the texture space gradients to obtain adjusted gradient values.

22. The method of claim 13, wherein the pixel shader calculates the texture space gradients from the texture coordinates and passes them with the texture coordinates to the texture unit, wherein the pixel shader indicates to the texture unit that the texture unit must correct the texture gradient values for any non-orthonormality of the sample grid.

23. The method of claim 22 wherein the pixel shader also determines the gradient scale factors and provides them to the texture unit, wherein the pixel shader notifies the texture unit that the texture unit must apply the gradient scale factors to the gradients to obtain the adjusted gradients.

24. The method of claim 13, wherein the pixel shader calculates the adjusted texture gradients from the texture coordinates and passes them along with the texture space coordinates to the texture unit, wherein the pixel shader indicates to the texture unit that the adjusted gradients should be used as is to select a level of detail for a texture to be applied to a primitive.

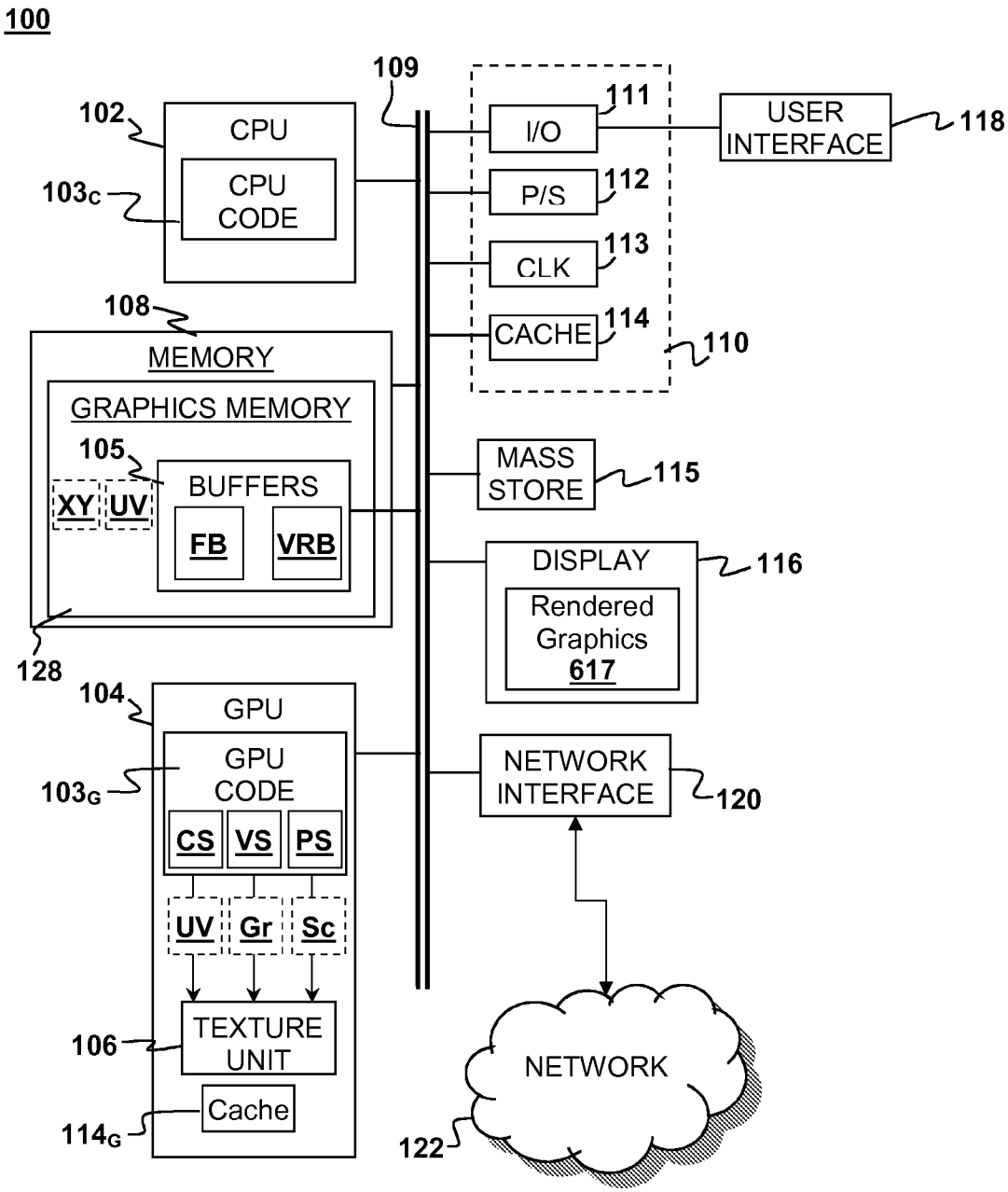

100
109
102
CPU
103C
CPU CODE
111
I/O
USER INTERFACE
118
112
P/S
113
CLK
114
CACHE
110
108
MEMORY
GRAPHICS MEMORY
105
BUFFERS
XY
UV
FB
VRB
MASS STORE
115
DISPLAY
116
Rendered Graphics 617
128
104
GPU
GPU CODE
103G
CS
VS
PS
UV
Gr
Sc
NETWORK INTERFACE
120
TEXTURE UNIT
106
Cache
114G
NETWORK
122

FIG. 1A

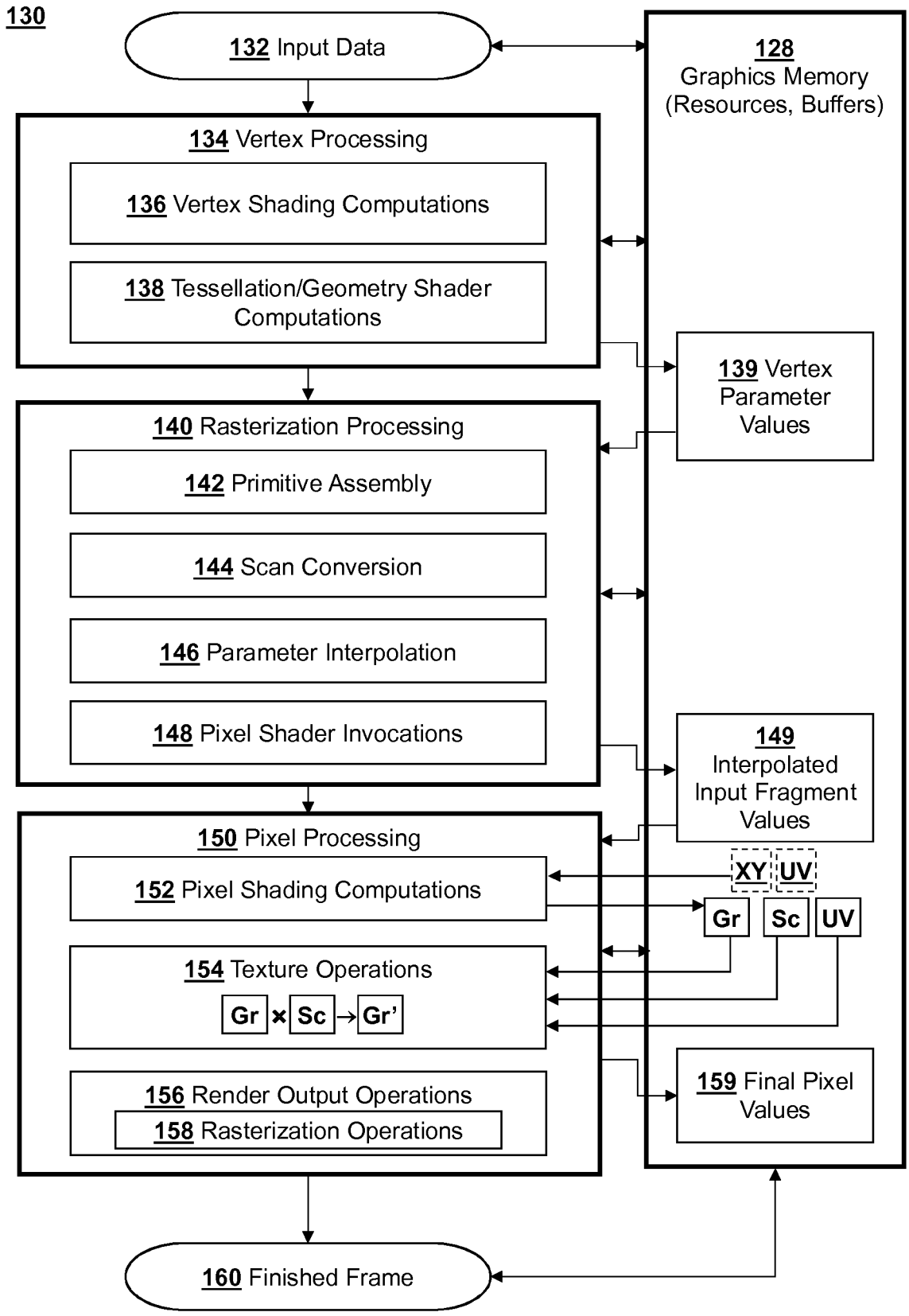

130
132 Input Data
128 Graphics Memory (Resources, Buffers)
134 Vertex Processing
136 Vertex Shading Computations
138 Tessellation/Geometry Shader Computations
139 Vertex Parameter Values
140 Rasterization Processing
142 Primitive Assembly
144 Scan Conversion
146 Parameter Interpolation
148 Pixel Shader Invocations
149 Interpolated Input Fragment Values
150 Pixel Processing
152 Pixel Shading Computations
XY
UV
Gr
Sc
UV
154 Texture Operations
Gr × Sc → Gr'
156 Render Output Operations
158 Rasterization Operations
159 Final Pixel Values
160 Finished Frame

FIG. 1B

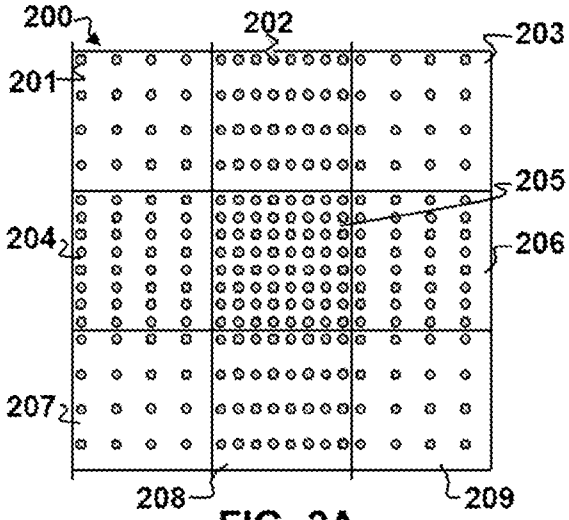


FIG. 2A

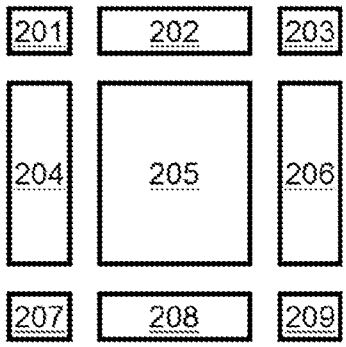


FIG. 2B

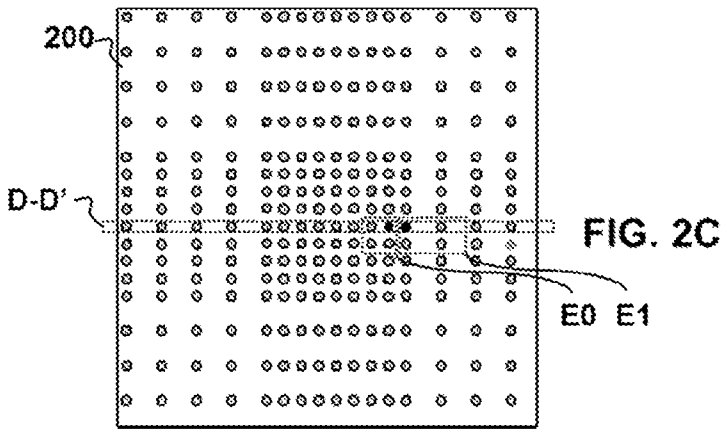


FIG. 2C

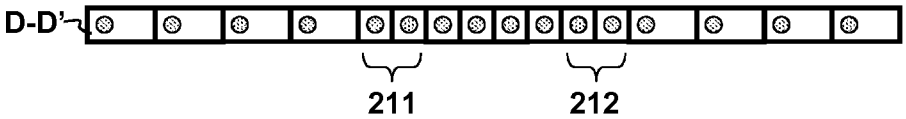


FIG. 2D

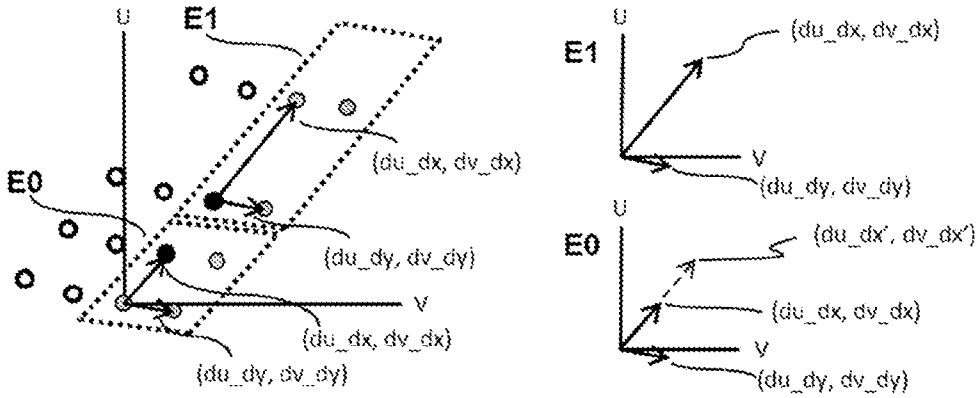

U
E1
E0
V
(du_dx, dv_dx)
(du_dy, dv_dy)
(du_dx, dv_dx)
(du_dy, dv_dy)
U
E1
V
(du_dx, dv_dx)
(du_dy, dv_dy)
U
E0
V
(du_dx', dv_dx')
(du_dx, dv_dx)
(du_dy, dv_dy)

FIG. 2E

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 15/21987

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G09G 5/00 (2015.01)
CPC - G06T 15/04; G06T 11/001; G06T 15/005
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8): G09G 5/00 (2015.01)
CPC: G06T 15/04; G06T 11/001; G06T 15/005

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 345/582; 345/418; 345/698 (Keyword limited; terms below); IPC(8): G09G 5/00 (2015.01) (Keyword limited; terms below); CPC: G06T 15/04; G06T 11/001; G06T 15/005; G06T 1/60; G09G 5/363 (Keyword limited; terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PatBase; Google (Scholar, Patents, Web)
Terms used: non-orthonormal grid gpu "pixel shader" texture unit calculate per-pixel space gradient smooth

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2009/0141033 A1 (STREET), 04 June 2009 (04.06.2009), entire document, especially Abstract; para [0002], [0005], [0032], [0034], [0037]-[0038], [0040], [0132]	1-24
Y	US 6,469,700 B1 (MUNSHI et al.), 22 October 2002 (22.10.2002), entire document, especially Abstract; Claim 1; col 2, ln 10-25; col 3, ln 28-46; col 5, ln 49-63; col 6, ln 57 to col 7, ln 5	1-24
Y	US 2013/0300740 A1 (SNYDER et al.), 14 November 2013 (14.11.2013), entire document, especially Abstract; para [0196]	2, 10-11, 14, 22-23

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 May 2015 (29.05.2015)	01 JUL 2015
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (January 2015)