

GPU RENDERING TO SYSTEM MEMORY

CROSS-REFERENCES TO RELATED APPLICATIONS

[0001] This application is related to U.S. Patent No. 6,275,243, entitled "Method and
5 apparatus for accelerating the transfer of graphical images" and issued August 14, 2001, and
the disclosure of this patent is incorporated by reference herein for all purposes.

BACKGROUND OF THE INVENTION

[0002] The present invention relates to the field of computer graphics. Many computer
10 graphic images are created by mathematically modeling the interaction of light with a three
dimensional scene from a given viewpoint. This process, called rendering, generates a two-
dimensional image of the scene from the given viewpoint, and is analogous to taking a
photograph of a real-world scene.

[0003] As the demand for computer graphics, and in particular for real-time computer
15 graphics, has increased, computer systems with graphics processing subsystems adapted to
accelerate the rendering process have become widespread. In these computer systems, the
rendering process is divided between a computer's general purpose central processing unit
(CPU) and the graphics processing subsystem. Typically, the CPU performs high level
operations, such as determining the position, motion, and collision of objects in a given
20 scene. From these high level operations, the CPU generates a set of rendering commands and
data defining the desired rendered image or images. For example, rendering commands and
data can define scene geometry, lighting, shading, texturing, motion, and/or camera
parameters for a scene. The graphics processing subsystem creates one or more rendered
images from the set of rendering commands and data.

[0004] A typical graphics processing subsystem includes one or more graphics processing
25 units (GPUs) or coprocessors. Each GPU executes rendering commands generated by the
CPU. In addition to one or more GPUs, a graphics processing subsystem also includes
memory. The graphics subsystem memory is used to store one or more rendered images to
be output to a display device, geometry data, texture data, lighting and shading data, and
30 other data used to produce one or more rendered images.

[0005] To maximize rendering performance, the graphics subsystem memory is typically segregated from the general purpose system memory used by the computer system. This allows the graphics processing subsystem to maximize memory access performance, and consequently, rendering performance. However, having separate memory for the graphics processing subsystem increases costs significantly, not only because of the expense of extra memory, which can be hundreds of megabytes or more, but also due to the costs of supporting components such as power regulators, filters, and cooling devices and the added complexity of circuit boards. Moreover, the extra space required for separate graphics processing subsystem memory can present difficulties, especially with notebook computers or mobile devices.

[0006] One solution to the problems associated with separate graphics processing subsystem memory is to use a unified memory architecture, in which all of the data needed by the graphics processing subsystem, for example geometry data, texture data, lighting and shading data, and rendered images, is stored in the general purpose system memory of the computer system. Traditionally, the data bus connecting the graphics processing subsystem with system memory limits the performance of unified memory architecture systems.

[0007] Improved data bus standards, such as the PCI-Express data bus standard, increase the bandwidth available for accessing memory; however, achieving optimal rendering performance with an unified memory architecture still requires careful attention to memory bandwidth and latency. Moreover, the PCI-Express data bus standard introduces its own problems, including system deadlock and high overhead for selective memory accesses. Additionally, scanout, which is the process of transferring a rendered image from memory to a display device, requires precise timing to prevent visual discontinuities and errors. Because of this, performing scanout from a rendered image stored in system memory is difficult.

[0008] It is therefore desirable for a graphics processing subsystem using a unified memory architecture to provide good rendering performance and error-free scanout from system memory. Moreover, it is desirable for the graphics processing subsystem to prevent problems such as system deadlock and high overhead for selective memory accesses.

BRIEF SUMMARY OF THE INVENTION

[0009] An embodiment of the invention enables a graphics processing subsystem to use system memory as its graphics memory for rendering and scanout of images. To prevent deadlock of the data bus, the graphics processing subsystem may use an alternate virtual channel of the data bus to access additional data from system memory needed to complete a write operation of a first data. In communicating with the system memory, a data packet including extended byte enable information allows the graphics processing subsystem to write large quantities of data with arbitrary byte masking to system memory. To leverage the high degree of two-dimensional locality of rendered image data, the graphics processing subsystem arranges image data in a tiled format in system memory. A tile translation unit converts image data virtual addresses to corresponding system memory addresses. The graphics processing subsystem reads image data from system memory and converts it into a display signal.

[0010] In an embodiment, a graphics processing subsystem comprises a rendering unit adapted to create image data for a rendered image in response to rendering data, and a data bus interface adapted to be connected with a system memory device of a computer system via a data bus. In response to a write operation of a first data to a graphics memory associated with the graphics processing subsystem, the graphics processing subsystem is adapted to retrieve a second data necessary to complete the write operation of the first data. The graphics processing subsystem then determines from the second data a destination for the first data in the system memory and redirects the write operation of the first data to the destination for the first data in the system memory. In a further embodiment, the destination for the first data in the system memory is within a portion of the system memory designated as the graphics memory associated with the graphics processing subsystem. In another embodiment, the second data includes address translation information, and the graphics processing subsystem is adapted to translate a virtual address associated with the graphics memory to a corresponding destination in system memory.

[0011] In an embodiment, the graphics processing subsystem is adapted to receive the write operation of the first data via the data bus interface from a first virtual channel of the data bus and to retrieve the second data from system memory via the data bus interface using a second virtual channel of the data bus. In an alternate embodiment, the graphics processing subsystem is adapted to retrieve the second data from a local memory connected with the graphics processing subsystem.

[0012] In a further embodiment, the graphics processing subsystem includes a tile address translation unit adapted to convert a virtual memory address corresponding to a location in an image to a memory address within a tiled arrangement of image data in system memory. The tile address translation unit may be further adapted to initiate a plurality of system memory accesses via the data bus interface over the data bus in response to a range of virtual memory addresses corresponding to a contiguous portion of an image. Depending upon the range of virtual memory addresses, the plurality of system memory accesses may be for non-contiguous portions of system memory.

[0013] In still another embodiment, the data bus interface is adapted to communicate a third data with the system memory via the data bus using a data packet of a first data packet type in response to an instruction indicating that a memory controller associated with the system memory is compatible with the first data packet type. The first data packet type includes extended byte enable data. In response to an instruction indicating that the memory controller is incompatible with the first data packet type, the data bus interface communicates the third data with the system memory via the data bus using a plurality of data packets of a second data packet type.

[0014] In an additional embodiment, the graphics processing subsystem includes a display device controller adapted to communicate a display signal corresponding with the rendered image with a display device. In one embodiment, the display device controller is adapted to retrieve image data corresponding with the rendered image from a local memory connected with the graphics processing subsystem. In another embodiment, the display device controller is adapted to retrieve image data corresponding with the rendered image from the system memory.

[0015] In an embodiment, the display device controller is adapted to retrieve a first image data corresponding with a first row of the rendered image from a tiled arrangement of image data in the system memory and to communicate the first image data with the display device. The graphics processing subsystem may retrieve a set of image data from the system memory corresponding with a set of tiles of the rendered image including the first row of the rendered image. The graphics processing subsystem may discard a portion of the set of image data not including the first row of image data. In an alternate embodiment, the display device controller includes an image data cache adapted to store a second image data included in the set of tiles and corresponding with at least one additional row of the rendered image. The

display device controller is adapted to retrieve the second image data from the image data cache subsequent to retrieving the first image data and to communicate the second image data with the display device.

5

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] The invention will be described with reference to the drawings, in which:

Figure 1 is a block diagram of a computer system suitable for practicing an embodiment of the invention;

10

Figure 2 illustrates a general technique for preventing system deadlock according to an embodiment of the invention;

Figure 3 illustrates a general technique for preventing system deadlock according to another embodiment of the invention;

Figures 4A and 4B illustrate a system for selectively accessing memory over a data bus according to an embodiment of the invention;

15

Figures 5A and 5B illustrate a system of organizing display information in system memory to improve rendering performance according to an embodiment of the invention;

Figure 6A and 6B illustrate a system for accessing display information according to an embodiment of the invention; and

20

Figures 7A-7C illustrate systems for outputting display information in system memory to a display device according to embodiments of the invention.

In the drawings, the use of identical reference numbers indicates identical components.

DETAILED DESCRIPTION OF THE INVENTION

25

[0017] Figure 1 is a block diagram of a computer system 100, such as a personal computer, video game console, personal digital assistant, or other digital device, suitable for practicing an embodiment of the invention. Computer system 100 includes a central processing unit (CPU) 105 for running software applications and optionally an operating system. In an embodiment, CPU 105 is actually several separate central processing units operating in parallel. Memory 110 stores applications and data for use by the CPU 105. Storage 115 provides non-volatile storage for applications and data and may include fixed disk drives,

30

removable disk drives, flash memory devices, and CD-ROM, DVD-ROM, or other optical storage devices. User input devices 120 communicate user inputs from one or more users to the computer system 100 and may include keyboards, mice, joysticks, touch screens, and/or microphones. Network interface 125 allows computer system 100 to communicate with other computer systems via an electronic communications network, and may include wired or wireless communication over local area networks and wide area networks such as the Internet. The components of computer system 100, including CPU 105, memory 110, data storage 115, user input devices 120, and network interface 125, are connected via one or more data buses 160. Examples of data buses include ISA, PCI, AGP, PCI, PCI-Express, and HyperTransport data buses.

[0018] A graphics subsystem 130 is further connected with data bus 160 and the components of the computer system 100. The graphics subsystem may be integrated with the computer system motherboard or on a separate circuit board fixedly or removably connected with the computer system. The graphics subsystem 130 includes a graphics processing unit (GPU) 135 and graphics memory. Graphics memory includes a display memory 140 (e.g., a frame buffer) used for storing pixel data for each pixel of an output image. Pixel data can be provided to display memory 140 directly from the CPU 105. Alternatively, CPU 105 provides the GPU 135 with data and/or commands defining the desired output images, from which the GPU 135 generates the pixel data of one or more output images. The data and/or commands defining the desired output images is stored in additional memory 145. In an embodiment, the GPU 135 generates pixel data for output images from rendering commands and data defining the geometry, lighting, shading, texturing, motion, and/or camera parameters for a scene.

[0019] In another embodiment, display memory 140 and/or additional memory 145 are part of memory 110 and is shared with the CPU 105. Alternatively, display memory 140 and/or additional memory 145 is one or more separate memories provided for the exclusive use of the graphics subsystem 130. The graphics subsystem 130 periodically outputs pixel data for an image from display memory 218 and displayed on display device 150. Display device 150 is any device capable of displaying visual information in response to a signal from the computer system 100, including CRT, LCD, plasma, and OLED displays. Computer system 100 can provide the display device 150 with an analog or digital signal.

[0020] In a further embodiment, graphics processing subsystem 130 includes one or more additional GPUs 155, similar to GPU 135. In an even further embodiment, graphics processing subsystem 130 includes a graphics coprocessor 165. Graphics processing coprocessor 165 and additional GPUs 155 are adapted to operate in parallel with GPU 135, or in place of GPU 135. Additional GPUs 155 generate pixel data for output images from rendering commands, similar to GPU 135. Additional GPUs 155 can operate in conjunction with GPU 135 to simultaneously generate pixel data for different portions of an output image, or to simultaneously generate pixel data for different output images. In an embodiment, graphics coprocessor 165 performs rendering related tasks such as geometry transformation, shader computations, and backface culling operations for GPU 135 and/or additional GPUs 155.

[0021] Additional GPUs 155 can be located on the same circuit board as GPU 135 and sharing a connection with GPU 135 to data bus 160, or can be located on additional circuit boards separately connected with data bus 160. Additional GPUs 155 can also be integrated into the same module or chip package as GPU 135. Additional GPUs 155 can have their own display and additional memory, similar to display memory 140 and additional memory 145, or can share memories 140 and 145 with GPU 135. In an embodiment, the graphics coprocessor 165 is integrated with the computer system chipset (not shown), such as with the Northbridge or Southbridge chip used to control the data bus 160.

[0022] System deadlock is a halt in system execution due to two operations both requiring a response from the other before completing their respective operations. One source of system deadlock results from posted write operations over the data bus connecting the CPU with the graphics processing subsystem. For many types of data buses, such as the PCI-Express data bus, a posted write operation must be completed before any other read or write operations can be performed over the data bus. A posted write operation is a write operation that is considered completed by the requester as soon as the destination accepts it. For performance purposes, PCI-Express uses posted write operations for all memory writes. PCI-Express write operations to configuration and I/O are typically non-posted write operations and require a confirmation that the write was finished, for example a completion message without data.

[0023] Because posted write operations block other data bus operations, a system deadlock can be created when the graphics processing subsystem requires additional information to

complete a posted write issued by a CPU. For example, graphics processing subsystems often utilize a linear contiguous memory organization for processing convenience; however, the portion of the general purpose memory of the computer system used as the graphics memory in a unified memory architecture is typically arranged as sets of non-contiguous memory pages. A page translation table translates memory addresses from the linear contiguous address space used by the graphics processing subsystem to the paged, non-contiguous address space used by the computer system memory, enabling the graphics processing subsystem to access the computer system memory. In an embodiment, the page translation table is stored in system memory. When the graphics processing subsystem needs to access a given memory address in the graphics memory, an address translation portion of the graphics processing subsystem retrieves the appropriate portion of the page translation table, translates the linear memory address used by the graphics processing subsystem to a corresponding paged memory address used by system memory, and then access the system memory at the translated memory address.

[0024] When the CPU issues a posted write command requesting data to be written to a destination in graphics memory, no other data bus completion operations can be completed until the posted write command is accepted. However, in order to accept the posted write command, the graphics processing subsystem must access the page translation table stored in system memory to determine the paged memory address corresponding to the destination in graphics memory. Because the posted write command blocks any subsequent bus completion operations until it has been accepted, the bus completion operation returning the data from the page translation table is blocked by the posted write operation. As the posted write operation cannot be completed until the graphics processing subsystem receives the data from the page translation table, the computer system is deadlocked.

[0025] In another example, graphics processing system may execute in several different contexts. A separate context may be associated with each application, window, and/or execution thread processed by the graphics processing subsystem. When switching between contexts, the graphics processing subsystem finishes all operations from the prior context, stores state information associated with the prior context, loads state information for the new context, and begins execution of operations for the new context. To ensure proper execution, commands from the new context cannot be executed until the state information for the new context is loaded from memory.

[0026] In a unified memory architecture system, context state information may be stored in system memory. When the CPU instructs the graphics processing system to switch contexts via a posted write operation, the graphics processing subsystem must load the new context state information from system memory. However, because the posted write operation is not completed until the graphics processing subsystem has switched contexts, the graphics processing subsystem cannot access the system memory via the data bus. Thus, the computer system becomes deadlocked.

[0027] Figure 2 illustrates a general technique for preventing deadlock in a computer system 200 according to an embodiment of the invention. A CPU 205 communicates a first data 210 with the graphics processing subsystem 220 via a posted write operation 215 over a data bus. To complete the posted write operation 215, the graphics processing subsystem 220 must retrieve second data 225 from system memory 230. Under previous techniques for accessing system memory 230, the posted write operation 215 would block the graphics processing subsystem 220 from accessing the second data 225 in system data.

[0028] In an embodiment of the invention, the graphics processing subsystem 220 accesses second data 225 in system memory 230 by opening an alternate virtual channel for communication over the data bus. Virtual channels provide independent paths for data communications over the same data bus. As an example, the PCI-Express bus specification allows one PCI-Express data bus to communicate several different exchanges of data, with each exchange of data occurring independently and without interference from the other exchanges of data on the PCI-Express data bus.

[0029] In response to the posted write operation 215 communicated to the graphics processing subsystem 220 via a first virtual channel, for example VC0, on the data bus, the graphics processing subsystem 220 opens a second virtual channel, for example VC1, on the data bus for retrieving the second data 225 from system memory 230. Using the second virtual channel, VC1, the graphics processing subsystem 220 sends a request 235 for the second data 225 to the system memory 230 over the data bus. A response 240 from the system memory 230 using the second virtual channel, VC2, of the data bus returns the second data 225 to the graphics processing subsystem 220.

[0030] Using the second data 225 retrieved via the second virtual channel from the system memory 230, the graphics processing subsystem determines the information needed to complete the posted write operation 215 of the first data 210. For example, if the posted

write operation 215 is attempting to write the first data 210 to the graphics memory, then the second data 225 may be a portion of an address translation table used to translate a linear memory address in graphics memory to a corresponding paged memory address in system memory 230. The graphics processing subsystem 220 may then write 245 the first data 210 to the graphics memory 250, which in the unified memory architecture of computer system 200 is located in a portion of the system memory 230. This completes the posted write operation 215 and frees the first virtual channel for additional operations.

[0031] In another example, the first data 210 may include a context switch command for the graphics processing subsystem 220. In this example, the CPU 205 communicates the first data 210 including the context switch command via a posted write operation 215 over a first virtual channel to the graphics processing subsystem 220. In response, the graphics processing subsystem 220 opens an alternate virtual channel, for example VC1, to retrieve the second data 225, which in this example includes context state information, from the system memory 230. The second data 225 is then used by the graphics processing subsystem 220 to switch contexts in accordance with the context switch command included in the first data 210, thereby completing the posted write operation 215 and freeing the first virtual channel for additional operations. The graphics processing subsystem also writes the context state information for the prior context to memory, so that it can be retrieved later when execution switches back to that context.

[0032] Figure 3 illustrates a general technique for preventing system deadlock in computer system 300 according to another embodiment of the invention. A CPU 305 communicates a first data 310 with the graphics processing subsystem 320 via a posted write operation 315 over a data bus. As in the embodiments discussed above, graphics processing subsystem 320 needs to retrieve a second data 325 to complete the posted write operation 315.

[0033] To avoid the problem of deadlock occurring when trying to retrieve the second data over a data bus blocked by the posted write operation 315, the embodiment of computer system 300 includes a local memory 330. In an embodiment, the local memory 330 is communicates with the graphics processing subsystem 320 via a separate data bus. The local memory 330 stores the second data 325 required by the graphics processing subsystem to complete a posted write operation 315. As the local memory 330 needs to store the second data 325 used to complete a posted write operation involving the first data 310, the local

memory 330 may be a small quantity of memory, thereby preserving many of the advantages of a unified memory architecture.

[0034] In response to the posted write operation 315 communicated to the graphics processing subsystem 320 via a first data bus between the CPU 305 and the graphics processing subsystem, the graphics processing subsystem 320 sends a request 335 for the second data 325 to the local memory 330 over a separate data bus. A response 340 from the local memory 330 returns the second data 325 via the separate data bus.

[0035] Using the second data 325 retrieved from the local memory 330, the graphics processing subsystem 320 determines the information needed to complete the posted write operation 315 of the first data 310. For example, if the posted write operation 315 is attempting to write the first data 310 to the graphics memory, then the second data 325 may be a portion of an address translation table used to translate a linear memory address in graphics memory to a corresponding paged memory address in system memory 355. The graphics processing subsystem 320 may then write 345 the first data 310 to the graphics memory 350, which in the unified memory architecture of computer system 300 is located in a portion of the system memory 355. This completes the posted write operation 315 and frees the data bus between the CPU 305 and the graphics processing subsystem 320 for additional operations.

[0036] In another example, the first data 310 may include a context switch command for the graphics processing subsystem 320. In this example, the CPU 305 communicates the first data 310 including the context switch command via a posted write operation 315 to the graphics processing subsystem 320. In response, the graphics processing subsystem 320 retrieves the second data 325, which in this example includes context state information, from the local memory 330. The second data 325 is then used by the graphics processing subsystem 320 to switch contexts in accordance with the context switch command included in the first data 310, thereby completing the posted write operation 315 and freeing the data bus between the CPU 305 and the graphics processing subsystem 320 for additional operations.

[0037] Another problem with implementing graphics processing subsystems with unified memory architectures is the high overhead associated with selective memory access. Graphics processing subsystems commonly write or update image data for a small number of sparsely distributed pixels, rather than a large, contiguous block of image data. Conversely,

computer systems optimize their system memory to be accessed in large, contiguous blocks. To cope with these differences, many data bus standards, such as PCI-Express, allow for the inclusion of byte enable data in addition to the data being written to memory. The byte enable data masks off portions of a block of data from being written to system memory.

5 Using byte enable data allows devices to send large blocks of contiguous data to system memory while only updating a small number of sparsely distributed pixels.

[0038] Despite the inclusion of byte enable data for selectively masking portions of a block of data from being written to memory, selective memory access still requires substantial overhead. As an example, figure 4A illustrates a standard packet 400 of data formatted
10 according to the PCI-Express standard. Packet 400 includes a header 405 and a body 410. The body 410 contains the data to be communicated from one device, such as a graphics processing subsystem, to another device, such as a memory controller associated with a computer's system memory, over a data bus. The header 405 includes information to direct the packet 400 to its intended destination.

15 [0039] The header 405 of the standard packet 400 also includes byte enable data 415. According to the PCI-Express standard, the byte enable data 415 is an 8-bit mask value. The byte enable data 415 allows the first four bytes of data 420 and the last four bytes of data 425 in the body 410 to be selectively masked according to the value of the byte enable data 415. For example, if the first bit in the byte-enable data 415 is a "0," then the first byte in the body
20 410 will not be written to the destination device. Conversely, setting a bit in the byte enable data 415 to "1" will allow the corresponding byte in the body 410 to be written to the destination device. However, the byte-enable data 415 only enables selective masking of the first four bytes 420 and the last four bytes 425 of the body. According to the PCI-Express standard, the middle bytes of data 430 in body 410 must be written to the destination device
25 in their entirety.

[0040] Because the PCI-Express standard limits the byte enable data to an 8-bit value, the graphics processing subsystem is severely limited when trying to write arbitrarily masked image data. The graphics processing subsystem often requires the ability to selectively mask any byte in the packet body. However, when using the standard PCI-Express packet, the
30 graphics processing subsystem can only mask up to eight bytes at a time. Thus, in using the standard PCI-Express packet, the graphics processing subsystem is limited to packets with 8-byte bodies when arbitrary byte masking is required. Should the graphics processing

subsystem require arbitrary byte masking for larger groups of data, the graphics processing subsystem must break the block of data down into 8-byte or less portions and use a separate PCI-Express packet for each portion.

[0041] Using PCI-Express packets with 8-byte bodies is extremely wasteful of data bus bandwidth. As a typical PCI-Express header is 20-bytes long, sending 8-bytes of data in a packet body requires 28 bytes total. This wasteful overhead is exacerbated when the graphics processing subsystem needs arbitrary byte masking for larger groups of data. For example, sending a 32-byte group of data with arbitrary byte masking requires four separate standard PCI-Express packets, consuming a total of 112 bytes of bus bandwidth.

[0042] Figure 4B illustrates an improved PCI-Express packet 450 allowing for arbitrary byte masking for communications over a data bus according to an embodiment of the invention. The PCI-Express standard allows for the definition of vendor defined packets. Vendor defined packets can have a non-standard packet header, provided that the destination device is capable of interpreting the non-standard packet header. Figure 4B illustrates a vendor-defined packet 450 including a non-standard header 455 and a body 460. As in the standard PCI-Express packet, the body 460 contains the data to be communicated from one device to another device over a data bus. The header 455 includes information to direct the packet 450 to its intended destination.

[0043] The non-standard header 455 also includes extended byte enable data 465. In an embodiment, the extended byte enable data 465 includes sufficient bits to allow for arbitrary masking of any byte in the body 460. In a further embodiment, the number of bits in the extended byte enable data 465 is equal to the number of bytes in the body 465. In an example implementation, the extended byte enable data 465 is 32 bits, allowing for up to 32 bytes of data in the body 460 to be selectively masked.

[0044] Because the destination device must be able to properly interpret the non-standard header 455 of the packet 450, an embodiment of the invention uses a device driver associated with the graphics processing subsystem to detect whether the computer's system memory controller, which may be integrated with the computer system Northbridge or with the CPU, is compatible with the non-standard header 455 of packet 460. If the system memory controller is compatible, the graphics processing subsystem is instructed to use the format of packet 450 for selectively masking data written to system memory. Conversely, if the system memory controller is not compatible, the device driver instructs the graphics processing

subsystem to use the format of the standard PCI-Express packet 400 for selectively masking data written to system memory.

[0045] Figures 5A and 5B illustrate a system of organizing display information in system memory to improve rendering performance according to an embodiment of the invention.

Typically, a two-dimensional array of image data has been arranged in system or graphics memory as a series of rows or columns connected end to end. For example, a frame buffer in memory will start with all of the image data for the first row of pixels in an image, followed by all of the image data for the second row of pixels in the image, then all of the image data for the third row of pixels in an image, and so forth.

[0046] Although this arrangement of image data simplifies the conversion of two-dimensional coordinates in an image to a corresponding location for memory, it requires additional data bus bandwidth for unified memory architectures. Graphics processing subsystems access graphics memory with a high degree of two-dimensional locality. For example, a graphics processing system may simultaneously create image data for a given pixel and the nearby pixels, both in the same row and in nearby rows.

[0047] Continuing with this example, when using the linear memory arrangement described above, the distance in system memory between adjacent pixels in different rows will be hundreds or thousands of bytes apart. Because this distance is greater than the size allowed for data bus packets, especially in prior implementations where the size of byte enable data in packet headers limits the length of packet bodies, as discussed above, the graphics processing subsystem must send multiple small packets over the data bus to the system memory to write image data for a group of adjacent pixels. The data bus overhead incurred by using multiple small packets, rather than one large packet, decreases the performance and efficiency of the computer system.

[0048] Figures 5A and 5B illustrate a system of organizing display information in system memory to improve rendering performance according to an embodiment of the invention. To leverage the two-dimensional locality of image data generated by the graphics processing subsystem, an embodiment of the invention organizes image data as a set of tiles. Each tile includes image data for a two-dimensional array of pixels. Figure 5A illustrates a portion of an image 500. Image 500 is divided into a number of tiles, including tiles 505, 510, and 515. In the embodiment of Figure 5A, each tile includes a 4 by 4 array of pixels. However, square and non-square tiles having any number of pixels may be used in alternate embodiments. In

Figures 5A, each pixel is labeled with the pixel's row and column in the image 500. For example, tile 505 includes the first four pixels in each of the first four rows of the image 500.

[0049] Figure 5B illustrates image data 550 representing a portion of the image 500 arranged in system memory according to an embodiment of the invention. In this embodiment, image data for each tile is stored contiguously in system memory. For example, the portion 555 of system memory stores image data for all of the pixels in tile 505. In an embodiment, following the image data for tile 505 is a portion 560 of system memory that stores image data for all of the pixels in tile 510. Alternatively, the portion 560 of system memory may store image data for all of the pixels in tile 515.

[0050] By storing image data as a set of two-dimensional tiles of pixels, the distance in system memory between pixels in adjacent rows is reduced, particularly in the cases where both pixels reside in the same tile. As a result, the graphics processing subsystem is frequently able to write a set of nearby pixels to system memory using a single write operation over the data bus.

[0051] In a further embodiment, the arrangement of image data as tiles in system memory is hidden from the portions of the graphics processing system and/or the CPU. Instead, pixels are referenced by a virtual address in a frame buffer arranged scanline by scanline as discussed above. A tile address translation portion of the graphics processing subsystem translates memory access requests using the virtual address into one or more access requests to the tiled arrangement of image data stored in system memory.

[0052] Figure 6A and 6B illustrate an example for accessing image data according to an embodiment of the invention. Figure 6A illustrates a portion of an example image 600. Example image 600 includes tiles 605, 610, 615, and 620. Region 625 corresponds to an example set of pixels to be accessed by the graphics processing subsystem. In this example, region 625 covers portions of tiles 605 and 610. In an embodiment, region 625 is referenced using one or more virtual addresses.

[0053] To retrieve image data corresponding to the region 625, the graphics processing subsystem translates the one or more virtual memory addresses used to reference the region 625 into one or more system memory addresses. In an embodiment, a tile translation table is used to translate between virtual memory addresses and system memory addresses. The graphics processing subsystem then retrieves all or portions of one or more tiles containing the desired image data.

[0054] Figure 6B illustrates a portion of system memory 600 including image data corresponding to the region 625 discussed above. The portion of system memory 600 includes image data 605 corresponding to tile 605 and image data 610 corresponding to tile 610. Within image data 605, a subset of image data 615 corresponds to the portion of region 625 within tile 605. Similarly, a subset of image data 620 within image data 610 corresponds to the portion of region 625 within tile 610.

[0055] In an embodiment, the graphics processing subsystem identifies one or more tiles including the requested region of the image. The graphics processing subsystem then retrieves each of the identified tiles and discards image data outside of the requested region.

The remaining portions of image data are then assembled by the graphics processing subsystem into a contiguous set of image data corresponding with the requested region of the image. Alternatively, the graphics processing subsystem may retrieve only the required portions of each identified tile. In this embodiment, the graphics processing subsystem may retrieve image data corresponding with a contiguous region of an image using a number of non-contiguous memory accesses.

[0056] The graphics processing subsystem may access regions of the image stored in system memory in a tiled arrangement for a number of different purposes, including reading and writing image data to render an image. In one embodiment, the graphics processing subsystem transfers an image to be displayed into a local memory, such as local memory 330 discussed above, prior to scanout. An alternate embodiment of the invention allows the graphics processing subsystem to transfer a rendered image from system memory to a display device, a process referred to as scanout. Scanout typically requires that image data be communicated to the display device at precise time intervals. If the graphics processing subsystem is unable to communicate image data with display device at the proper time, for example, due to a delay in retrieving image data from system memory, visual artifacts such as tearing will be introduced.

[0057] Typically, image data is communicated row by row to a display device. In an embodiment, the graphics processing subsystem retrieves image data for one or more rows of the image ahead of the row being communicated to the display device. Figure 7A illustrates an example application of this embodiment. Figure 7A illustrates an image 700. In an embodiment, the image is divided into tiles as discussed above. The graphics processing subsystem communicates row 705 of image 700 to the display device. As row 705 is being

communicated with the display device, the graphics processing subsystem is retrieving image data from system memory for subsequent rows of image 700, for example row 710 and/or row 715.

[0058] Figure 7B illustrates the operation of an example portion 730 of the graphics processing subsystem used to communicate image data with a display device. Portion 730 includes a scanout unit 735 adapted to convert image data into a display signal to be communicated with a display device 740. The display signal output by the scanout unit 735 may be a digital or analog signal.

[0059] In an embodiment, the scanout unit retrieves image data in a tiled format from the system memory. Because each tile of image data includes two or more rows of image data for a portion of the image, the scanout unit 735 assembles the desired row of the image from portions of each retrieved tile. For example, the image data for row 705 may be assembled from portions of a number of different tiles, including portion 745 of image data 760 for tile 707, portion 750 of image data 765 for tile 709, and portion 755 of image data 770 for tile 711. In an embodiment, the unused portions of the retrieved tiles are discarded.

[0060] In an alternate embodiment, as the scanout unit 735 retrieves image data for a given row from a set of tiles, the scanout unit 735 also stores image data for one or more subsequent rows of the image. This reduces the number accesses to system memory for scanout, thereby improving the efficiency and performance of the graphics processing subsystem.

[0061] Figure 7C illustrates the operation of an example implementation 771 of this alternate embodiment. In this example, the scanout unit 736 retrieves tiles of image data from system memory that include the row desired by the scanout unit 736. The scanout unit assembles the desired row from appropriate portions of each retrieved tile. For example, image data for row 705 can be assembled from a number of portions of different tiles, including portions 745, 750, and 755.

[0062] As the image data for the desired row is assembled from portions of a number of tiles, image data for one or more subsequent rows is stored in one or more scanline caches. For example, the image data for the first row subsequent to the desired row, for example row 710, is stored in scanline cache 790. Image data for the first subsequent row may include tile portions 772, 774, and 776. Similarly, image data for the second subsequent row, including

tile portions 778 and 780, is stored in scanline cache 788. Image data for the third subsequent row, including tile portions 782, 783, and 784, is stored in scanline cache 785.

[0063] Thus, for subsequent rows, scanline unit 736 can retrieve image data for the next desired row from the appropriate scanline cache. In an embodiment, there is a scanline cache
5 corresponding to each row of image data in a tile of an image, so that the scanline unit 735 only needs to read each tile from system memory once for a given image. Alternate embodiments may have fewer scanline caches to reduce the hardware complexity of the graphics processing subsystem.

[0064] This invention provides a graphics processing subsystem capable of using system
10 memory as its graphics memory for rendering and scanout of images. Although this invention has been discussed with reference to computer graphics subsystems, the invention is applicable to other components of a computer system, including audio components and communications components. The invention has been discussed with respect to specific examples and embodiments thereof; however, these are merely illustrative, and not
15 restrictive, of the invention. Thus, the scope of the invention is to be determined solely by the claims.

WHAT IS CLAIMED IS:

1 1. A graphics processing subsystem, comprising:
2 a rendering unit adapted to create image data for a rendered image in response
3 to rendering data; and
4 a data bus interface adapted to be connected with a system memory device of a
5 computer system via a data bus;
6 wherein in response to a write operation of a first data to a graphics memory
7 associated with the graphics processing subsystem, the graphics processing subsystem is
8 adapted to retrieve a second data necessary to complete the write operation of the first data, to
9 determine from the second data a destination for the first data in the system memory, and to
10 redirect the write operation of the first data to the destination for the first data in the system
11 memory.

1 2. The graphics processing subsystem of claim 1, wherein the destination
2 for the first data in the system memory is within a portion of the system memory designated
3 as the graphics memory associated with the graphics processing subsystem.

1 3. The graphics processing subsystem of claim 1, further adapted to
2 receive the write operation of the first data via the data bus interface from a first virtual
3 channel of the data bus and to retrieve the second data from system memory via the data bus
4 interface using a second virtual channel of the data bus.

1 4. The graphics processing subsystem of claim 1, further adapted to
2 retrieve the second data from a local memory connected with the graphics processing
3 subsystem.

1 5. The graphics processing subsystem of claim 1, wherein the second data
2 includes address translation information, and the graphics processing subsystem is adapted to
3 translate a virtual address associated with the graphics memory to a corresponding destination
4 in system memory.

1 6. The graphics processing subsystem of claim 1, wherein the second data
2 includes context state information, and the graphics processing subsystem is adapted to
3 perform a context switch in response to the first data.

1 7. The graphics processing subsystem of claim 1, further comprising:
2 a tile address translation unit adapted to convert a virtual memory address
3 corresponding to a location in an image to a memory address within a tiled arrangement of
4 image data in system memory.

1 8. The graphics processing subsystem of claim 7, wherein the tile address
2 translation unit is further adapted to initiate a plurality of system memory accesses via the
3 data bus interface over the data bus in response to a range of virtual memory addresses
4 corresponding to a contiguous portion of an image.

1 9. The graphics processing subsystem of claim 8, wherein the plurality of
2 system memory accesses are for non-contiguous portions of system memory.

1 10. The graphics processing subsystem of claim 1, wherein the data bus
2 interface is adapted to communicate a third data with the system memory via the data bus
3 using a data packet of a first data packet type in response to an instruction indicating that a
4 memory controller associated with the system memory is compatible with the first data
5 packet type and to communicate the third data with the system memory via the data bus using
6 a plurality of data packets of a second data packet type in response to an instruction
7 indicating that the memory controller is incompatible with the first data packet type.

1 11. The graphics processing subsystem of claim 10, wherein the first data
2 packet type includes extended byte enable data.

1 12. The graphics processing subsystem of claim 1, further including a
2 display device controller adapted to communicate a display signal corresponding with the
3 rendered image with a display device.

1 13. The graphics processing subsystem of claim 12, wherein the display
2 device controller is adapted to retrieve image data corresponding with the rendered image
3 from a local memory connected with the graphics processing subsystem.

1 14. The graphics processing subsystem of claim 12, wherein the display
2 device controller is adapted to retrieve image data corresponding with the rendered image
3 from the system memory.

1 15. The graphics processing subsystem of claim 14, wherein the display
2 device controller is adapted to retrieve a first image data corresponding with a first row of the
3 rendered image from a tiled arrangement of image data in the system memory and to
4 communicate the first image data with the display device.

1 16. The graphics processing subsystem of claim 15, wherein the display
2 device controller is adapted to retrieve a set of image data from the system memory
3 corresponding with a set of tiles of the rendered image including the first row of the rendered
4 image.

1 17. The graphics processing subsystem of claim 16, wherein the display
2 device controller is adapted to discard a portion of the set of image data not including the first
3 row of image data.

1 18. The graphics processing subsystem of claim 16, wherein the display
2 device controller includes an image data cache adapted to store a second image data included
3 in the set of tiles and corresponding with at least one additional row of the rendered image;
4 and

5 wherein the display device controller is adapted to retrieve the second image
6 data from the image data cache subsequent to retrieving the first image data and to
7 communicate the second image data with the display device.

1 19. A graphics processing subsystem, comprising:
2 a display device controller adapted to retrieve a first image data corresponding
3 with a first row of a rendered image from a tiled arrangement of image data in a system
4 memory and to communicate the first image data with a display device.

1 20. The graphics processing subsystem of claim 19, wherein the display
2 device controller is adapted to discard a portion of the set of image data not including the first
3 row of image data.

1 21. The graphics processing subsystem of claim 19, wherein the display
2 device controller includes an image data cache adapted to store a second image data included
3 in the set of tiles and corresponding with at least one additional row of the rendered image;
4 and

5 wherein the display device controller is adapted to retrieve the second image
6 data from the image data cache subsequent to retrieving the first image data and to
7 communicate the second image data with the display device.