

TILED GRAPHICS ARCHITECTURE

Field Of The Invention

The present invention pertains to the field of computer systems. More particularly,
5 this invention pertains to the field of reducing primitive storage requirements and
improving memory bandwidth utilization in a tiled graphics architecture.

Background of the Invention

In typical computer graphics systems, a three dimensional (3D) object to be
10 represented on the display screen is composed of graphics primitives such as triangle lists,
triangle strips and triangle fans etc. Typically, the primitives of a 3D object to be rendered
are defined by a host computer in terms of primitive data. For example, for each triangle in
a primitive, the host computer may define the three vertices of the triangle in terms of its
spatial location in terms of X, Y and Z coordinates, as well as data defining the red, green
15 and blue (R, G and B) color values of each vertex, and texture coordinates. Additional
primitive data may be used in specific applications. Rendering hardware within a graphics
controller interpolates the primitive data to compute the display screen pixels that
represent each primitive, and the R, G and B color values for each pixel.

In order to make more efficient use of memory bandwidth, graphics primitives are
20 sorted into bins, also referred to as "tiles". This well-know technique is often referred to
as "tiling".

Figure 1 and Figure 2 show an example of sorting graphics primitives into bins, or
tiles. For this example, a microprocessor fetches data for primitives 110, 120, and 130
from a primitive storage area. The primitive storage area may be implemented as a portion
25 of the main system memory or may be implemented as a local graphics memory directly

coupled to the graphics controller. The primitives 110, 120, and 130 are eventually to be rendered and then displayed on a display screen, represented by block 100. The block 100 is divided into four bins for this example. Typically, a frame of display data is divided into many more bins than the four shown in this example with a typical bin dimension of 128 x 64 pixels. Four bins are used in this example in order to simplify the discussion.

After fetching data for a graphics primitive, the processor determines which bin or "tile" the primitive intersects. For example, the processor can determine that primitive 110 intersects bin 210 as well as bin 220. The processor then writes the data for the three vertices of primitive 110 to an area of graphics memory set aside to store primitive data for bin 210 as well as to an area of graphics memory set aside to store primitive data for bin 220. Similarly, the processor writes vertex data for primitive 120 to storage areas for bins 220 and 240 and writes vertex data for primitive 130 to storage areas for bins 210, 230, and 240. Once the primitives have been sorted into bins, the graphics controller fetches primitive data from the graphics memory and renders the primitives one bin at a time.

Figure 2 demonstrates how the graphics controller divides the primitives 110, 120, and 130 into various primitives that fit into bins 210, 220, 230, and 240. The various primitives are divided into bins according to how the primitives intersect the bin boundaries. For example, when the primitive data for bin 210 is fetched from graphics memory, the graphics controller divides primitive 110 to create primitive 211. Primitive 130 is divided to create primitive 212. The graphics controller then proceeds to render primitives 211 and 212. The graphics controller then proceeds to process bin 220 by dividing primitives 110 and 120 to create primitives 221 and 222 and by rendering the

primitives 221 and 222. The graphics controller continues in a similar fashion to process bins 230 and 240.

Figure 3 is a block diagram of a prior computer system that implements a tiled graphics architecture. Figure 3 shows a processor 310, a system memory 330 including a graphics primitive storage area 332, a graphics controller 340, and a display monitor 350.

Prior tiled graphics architectures such as that implemented by the system of Figure 3 have a disadvantage of using large amounts of memory bandwidth when moving primitive data from device to device. For example, when the processor 310 processes a primitive, the processor 310 reads the vertex data for the primitive from the graphics primitive storage area 332. The processor 310 then determines which bins that the primitive intersects. The processor 310 then must write several copies of the vertex data back out to the graphics primitive storage area 332 where the number of copies to be written depends on how many bins the primitive intersects.

The impact on memory bandwidth utilization can be demonstrated by considering that a typical graphics primitive can be represented by about 100 bytes of vertex data and that a graphics primitive may intersect several bins. This example will assume that a typical primitive intersects 3 bins. In this situation, the processor 310 must write an average of 300 bytes of vertex data to the graphics primitive storage area 332 for each primitive processed. For one frame of a relatively simple display consisting of 2k graphics primitives, the processor 310 must deliver 600k bytes of data per frame. If the frame display rate is 60 frames per second, the processor 310 must deliver data to the graphics primitive storage area 332 at a rate of 360M bytes per second. For a more complicated display consisting of 100k primitives, the bandwidth requirements increase to 1.8G bytes

per second. The same bandwidth requirements must also be met between the graphics primitive storage area 332 and the graphics controller 340. This high utilization of memory bandwidth to move graphics primitive data from the processor 310 to the graphics primitive storage area 332 and from the graphics primitive storage area 332 to the graphics controller 340 can result in a significant negative impact on overall system performance.

Brief Description of the Drawings

The invention will be understood more fully from the detailed description given below and from the accompanying drawings of embodiments of the invention which, however, should not be taken to limit the invention to the specific embodiments described, but are for explanation and understanding only.

Figure 1 is a diagram of several 3D objects arranged on a display screen in accordance with prior systems.

Figure 2 is a diagram depicting the several 3D objects of Figure 1 sorted into bins in accordance with prior systems.

Figure 3 is a block diagram of a prior system including a tiled graphics architecture.

Figure 4 is a flow diagram of an embodiment of a method for reducing memory bandwidth utilization in a tiled graphics architecture.

Figure 5 is a flow diagram of an embodiment of a method for reducing memory bandwidth utilization in a tiled graphics architecture where the graphics primitive storage area is located in system memory.

Figure 6 is a flow diagram of an embodiment of a method for reducing memory bandwidth utilization in a tiled graphics architecture where the graphics primitive storage area is located in a local graphics memory.

Figure 7 is a block diagram of a system including an embodiment of a graphics controller that includes a vertex cache.

Detailed Description

An example embodiment of a method and apparatus for reducing memory bandwidth utilization in a tiled graphics architecture will be described. For this example, a
5 microprocessor reads vertex data for a graphics primitive from graphics memory. The processor determines which bins the graphics primitive intersects. All vertices of the primitive are written into a vertex buffer for future reference. The vertex buffer may reside in either main system memory or local graphics memory. The vertex buffer may be implemented as part of the bin storage area or in a separate memory location.

10 Assuming that the processor determines that the graphics primitive intersects a first and a second bin, the processor writes a pointer to the first and second bin storage areas. The pointer indicates the location in memory of the actual vertex data. Thus, only one copy of the vertex data is moved from the processor to the graphics memory. Because the pointer is smaller in size than the vertex data, less data is moved from the processor to the
15 graphics memory and memory bandwidth utilization is improved.

The microprocessor in the above example and in the example embodiments that follow may be substituted for a 3D graphics processor that handles the same primitive processing as performed by the microprocessor. For example, an additional embodiment may include a 3D graphics processor that performs hardware transformation and lighting
20 calculations in hardware.

The graphics memory in the above example and in the example embodiments that follow may be included as part of a main system memory or may be implemented as a local graphics memory directly coupled to a graphics controller.

The term "pointer" as used herein is meant to include any means of at least partially indicating the location of vertex data, including a memory address and also including an index. For example, the pointer may be a physical or virtual memory address indicating the location of the vertex data. Alternatively, the pointer may be an index that
5 can be used to calculate the address location of the vertex data. For example, an address may be calculated from an index according to the equation "base address + index * vertex data size".

Although the above example and the examples that follow discuss a given number of bins which a graphics primitive may intersect, other examples are possible using any
10 number of bins. Further, although the graphics primitives discussed herein comprise triangles including three vertices, other types of primitives are possible.

Also, for the example embodiments described herein, an address is assumed to be 32 bits wide, an index is assumed to be 16 bits wide, and vertex data for a triangular graphics primitive is assumed to be approximately 100 bytes long. Other embodiments are
15 possible using a wide range of address, index, and data sizes and lengths.

Figure 4 is a flow diagram of an embodiment of a method for improving memory bandwidth utilization in a tiled graphics architecture. At block 410, a determination is made as to whether a graphics primitive intersects a first and a second bin. If the graphics primitive is found to intersect the first and the second bin, then at block 420 data for a
20 plurality of vertices corresponding to the graphics primitive are written to a first bin storage area located in a memory device. The memory device may comprise the main system memory or may comprise a local graphics memory coupled directly to a graphics controller.

At block 430 a plurality of pointers are written to a second bin storage area located in the graphics memory. The plurality of pointers indicate the memory locations for the data for the plurality of vertices. By writing pointers to the second bin storage area instead of writing the vertex data, less data is moved from the processor to the graphics memory and memory bandwidth utilization is improved. The pointers will be fetched by the graphics controller along with any other second bin primitive data. The graphics controller will use the pointers to fetch the vertex data from the first bin storage area.

Figure 5 is a flow diagram of an embodiment of a method for improving memory bandwidth utilization in a tiled graphics architecture in a computer system where the graphics memory is implemented as an area within main system memory and where the graphics controller includes a vertex cache. The vertex cache provides temporary storage for vertex data and allows for an improvement in system memory to graphics controller memory bandwidth utilization by reducing the amount of vertex data moved between the graphics memory located in main system memory and the graphics controller.

Referring to Figure 5, at block 505 a processor fetches vertex data for a graphics primitive from system memory and at block 510 the processor performs calculations on the vertex data. For this example, the vertex data for the graphics primitive includes data for three vertices, although other embodiments are possible where the vertex data for the graphics primitive may include data for any number of vertices. The calculations described as part of this embodiment are meant to represent a broad range of well-known techniques for manipulating graphics primitive data.

At block 515, the processor determines whether the graphics primitive intersects a first bin, and, assuming there is an intersection, the processor writes the vertex data for the graphics primitive to a first bin storage area in system memory.

At block 520, the processor determines whether the graphics primitive intersects a
5 second bin. If the graphics primitive is found to intersect the second bin, then at block 525 the processor writes three pointers to a second bin storage area in system memory. The pointers indicate the memory locations of the three vertices that were previously written to system memory.

At block 530, the processor determines whether the graphics primitive intersects a
10 third bin. If the graphics primitive is found to intersect the third bin, then at block 535 the processor writes three pointers to a third bin storage area in system memory. The pointers indicate the memory locations of the three vertices that were previously written to system memory.

At block 540, the processor determines whether the graphics primitive intersects a
15 fourth bin. If the graphics primitive is found to intersect the fourth bin, then at block 545 the processor writes three pointers to a fourth bin storage area in system memory. The pointers indicate the memory locations of the three vertices that were previously written to system memory.

Although this embodiment describes a graphics primitive possibly intersecting four
20 bins, other embodiments are possible where the graphics primitive may possibly intersect two or more bins. Further, in one embodiment, a bin may have the dimensions of 128 pixels by 64 pixels, although other bin dimensions are possible. Also the bin intersection determination may be performed in a parallel manner instead of the serial approach

described above. For example, a bounding box of the primitive may be used to find all bins that the primitive intersects at once.

As indicated by block 547, blocks 505 through 545 may be repeated until all primitives have been sorted into bins.

5 At block 550, the graphics controller fetches data from the first bin storage area. The data fetched from the first bin storage area and the vertex buffer includes the vertex data for the graphics primitive that was previously written to the system memory at block 515.

10 At block 555 the graphics controller stores the fetched vertex data in the vertex cache. In one embodiment, the vertex cache includes 16 entries which are 4-way interleaved with each entry capable of storing 32 bytes of vertex data. Other embodiments are possible with different numbers of entries and numbers of ways and further with each entry capable of storing different amounts of vertex data.

15 After the graphics controller fetches the first bin data and stores the vertex data in the vertex cache, the graphics controller renders the first bin primitives at block 560. As part of the rendering process, the graphics controller determines which portion of each graphics primitive included in the first bin data falls within the first bin and renders only that portion of the primitive.

20 Following the rendering of the first bin, the graphics controller proceeds to process the second bin. As a first step in processing the second bin, the graphics controller fetches data from the second bin storage area at block 565. The data fetched from the second bin storage area includes pointers to the vertex data for the graphics primitive, assuming that an intersection with the second bin was found at block 520. At block 570, the graphics

controller uses the pointers to access the vertex data that was previously stored in the vertex cache at block 555. Once the graphics processor has accessed the vertex data, the graphics controller renders the second bin primitives at block 575.

At block 580, a determination is made as to whether additional bins remain to be rendered. If additional bins remain, then processing resumes at block 565. Blocks 565 through 580 are repeated until all bins have been rendered and processing stops at block 585. Note that the sequencing of bin rendering may or may not be serial. The above embodiment may be generalized to, based on certain heuristics, render the second bin first and followed by the third, first, and fourth bins. This allows overall system performance optimization measures. For example, load balancing can be used to normalize loadings on front-end and back-end processing in the graphics processor.

Figure 6 is a flow diagram of an embodiment of a method for improving memory bandwidth utilization in a tiled graphics architecture in a computer system where the graphics memory is implemented as a local graphics memory directly coupled to a graphics controller. The local graphics memory provides storage for vertex data and allows for an improvement in system memory to graphics controller memory bandwidth utilization by reducing the amount of vertex data moved between the graphics memory located in main system memory and the graphics controller.

Referring to Figure 6, at block 605 a processor fetches vertex data for a graphics primitive from local graphics memory or, alternatively, from system memory and at block 610 the processor performs calculations on the vertex data. For this example, the vertex data for the graphics primitive includes data for three vertices, although other embodiments are possible where the vertex data for the graphics primitive may include

data for any number of vertices. The calculations described as part of this embodiment are meant to represent a broad range of well-known techniques for manipulating graphics primitive data. At block 615, the processor determines whether the graphics primitive intersects a first bin, and, assuming there is an intersection, the processor writes the vertex
5 data for the graphics primitive to a first bin storage area in the local graphics memory.

At block 620, the processor determines whether the graphics primitive intersects a second bin. If the graphics primitive is found to intersect the second bin, then at block 625 the processor writes three pointers to a second bin storage area in local graphics memory. The pointers indicate the memory locations of the three vertices that were previously
10 written to local graphics memory.

At block 630, the processor determines whether the graphics primitive intersects a third bin. If the graphics primitive is found to intersect the third bin, then at block 635 the processor writes three pointers to a third bin storage area in local graphics memory. The pointers indicate the memory locations of the three vertices that were previously written to
15 local graphics memory.

At block 640, the processor determines whether the graphics primitive intersects a fourth bin. If the graphics primitive is found to intersect the fourth bin, then at block 645 the processor writes three pointers to a fourth bin storage area in local graphics memory. The pointers indicate the memory locations of the three vertices that were previously
20 written to local graphics memory.

Although this embodiment describes a graphics primitive possibly intersecting four bins, other embodiments are possible where the graphics primitive may possibly intersect two or more bins. Further, in one embodiment, a bin may have the dimensions of 128

pixels by 64 pixels, although other bin dimensions are possible. Also the bin intersection determination may be performed in a parallel manner instead of the serial approach described above. For example, a bounding box of the primitive may be used to find all bins that the primitive intersects at once.

5 As indicated by block 647, blocks 605 through 645 may be repeated until all primitives have been sorted into bins.

At block 650, the graphics controller fetches data from the first bin storage area. The data fetched from the first bin storage area includes the vertex data for the graphics primitive that was previously written to the local graphics memory at block 615.

10 After the graphics controller fetches the first bin data, the graphics controller renders the first bin primitives at block 660. As part of the rendering process, the graphics controller determines which portion of each graphics primitive included in the first bin data falls within the first bin and renders only that portion of the primitive.

Following the rendering of the first bin, the graphics controller proceeds to process
15 the second bin. As a first step in processing the second bin, the graphics controller fetches data from the second bin storage area at block 665. The data fetched from the second bin storage area includes pointers to the vertex data for the graphics primitive, assuming that an intersection with the second bin was found at block 620. At block 670, the graphics controller uses the pointers to access the vertex data that was previously stored in the local
20 graphics memory at block 615. Once the graphics processor has accessed the vertex data, the graphics controller renders the second bin primitives at block 675.

At block 680, a determination is made as to whether additional bins remain to be rendered. If additional bins remain, then processing resumes at block 665. Blocks 665

through 680 are repeated until all bins have been rendered and processing stops at block 685. Note that the sequencing of bin rendering may or may not be serial. The above method may be generalized to, based on certain heuristics, render second bin first and then followed by the third, first, and fourth bins. This allows overall system performance optimization measures. For example, load balancing can be used to normalize loadings on front-end and back-end processing in the graphics processor.

Figure 7 is a block diagram of a computer system including a graphics controller 740 that includes a vertex cache 742. The computer system of Figure 7 also includes a processor 710 coupled to a system logic device 720 via a processor bus 715. The system logic device 720 provides communication between the processor 710 and a system memory 730. The system memory 730 includes a graphics primitive storage area 732. The graphics primitive storage area 732 may be separated into storage areas for a plurality of bins.

The system logic device 720 also serves to couple the graphics controller 740 to the processor 710 and the system memory 730. The system of Figure 7 also includes a display monitor 750 coupled to the graphics controller 740.

The system of Figure 7 may be used with embodiments of methods for improving memory bandwidth utilization such as those discussed above in connection with Figures 4 and 5. For example, the processor 710 may read vertex data for a graphics primitive from the graphics primitive storage area 732. The processor 710 may then determine which bins the graphics primitive intersects. The processor 710 then writes the vertex data to a first bin storage area within the graphics primitive storage area 732. If the graphics primitive is found to intersect other bins, then the processor 710 writes pointers to other

bin storage areas within the graphics primitive storage area 732. The pointers indicate the location within the first bin storage area where the vertex data is stored. The pointers in this example include a 16 bit index from which the memory location of the vertex data may be calculated. Other embodiments are possible where the pointers include a 32 bit address identifying the storage locations of the vertex data. Still other embodiments are possible using different length indices and/or addresses.

When the graphics controller 740 desires to process the first bin, the graphics controller 740 fetches the first bin data from the graphics primitive storage area 732. The graphics controller 740 stores the vertex data for the graphics primitive in the vertex cache 742. The graphics controller 740 then renders the first bin, including the portion of the graphics primitive that falls within the first bin.

For this example, a bin has the dimensions of 128 x 64 pixels. The vertex cache 742 in this example includes 16 entries that are 4-way set-associative and are capable of storing 32 bytes of vertex data. The graphics primitive of this example is represented by three vertices where each vertex is defined by 32 bytes of data. Other embodiments are possible using other bin dimensions and/or other cache arrangements.

When the graphics controller 740 is ready to process the second bin, the graphics controller 740 fetches the data for the second bin from the graphics primitive storage area 732. The data for the second bin will include pointers to the vertex data for the graphics primitive assuming that the processor 710 previously determined that the graphics primitive intersects the second bin. The graphics controller 740 then uses the pointers to access the vertex data stored in the vertex cache 742. The vertex cache 742 serves to improve memory bandwidth utilization by eliminating the necessity for the vertex data to

be fetched from the graphics primitive storage area 732 when a copy of the vertex data is stored in the vertex cache 742, as is the case in this example.

Once the vertex data is retrieved from the vertex cache 742, the graphics controller 740 can render the second bin. Subsequent bins can be processed in a similar manner until
5 all bins have been rendered.

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

Reference in the specification to "an embodiment," "one embodiment," "some embodiments," or "other embodiments" means that a particular feature, structure, or characteristic described in connection with the embodiments is included in at least some embodiments, but not necessarily all embodiments, of the invention. The various
15 appearances of "an embodiment," "one embodiment," or "some embodiments" are not necessarily all referring to the same embodiments.

Claims

1. A method, comprising:
determining that a graphics primitive intersects a first tile and a second plurality of tiles;
writing data for a plurality of vertices corresponding to the graphics primitive to a first bin storage area located in a memory device; and
writing a plurality of pointers to a second plurality of bin storage areas located in the memory device, the plurality of pointers to indicate the location of the data for the plurality of vertices.
2. The method of claim 1, wherein writing data for a plurality of vertices corresponding to the graphics primitive to a first bin storage area located in a memory device includes writing data for a plurality of vertices corresponding to the graphics primitive to a first bin storage area located in a frame buffer.
3. The method of claim 2, wherein writing a plurality of pointers to a second plurality of bin storage areas located in the memory device

includes writing a plurality of pointers to a second plurality of bin storage areas located in the frame buffer.

4. The method of claim 1, wherein writing data for a plurality of vertices corresponding to the graphics primitive to a first bin storage area located in a memory device includes writing data for a plurality of vertices corresponding to the graphics primitive to a first bin storage area located in a main memory.

5. The method of claim 4, wherein writing a plurality of pointers to a second plurality of bin storage areas located in the memory device includes writing a plurality of pointers to a second plurality of bin storage areas located in the main memory.

6. The method of claim 5, further comprising:
loading data for one of the plurality of vertices into a vertex cache;
reading one of the plurality of pointers into a graphics controller; and
accessing the data for the one of the plurality of vertices stored in the vertex cache using the one of the plurality of pointers.

7. An apparatus comprising a bin fetch unit to fetch primitive data from a plurality of bin storage areas located in a memory, the primitive data including a plurality of pointers to indicate a memory location for data corresponding to a vertex, the bin fetch unit further to fetch the data corresponding to the vertex indicated by the plurality of pointers from an additional bin storage area.

8. The apparatus of claim 7, the memory including a main memory device.

9. The apparatus of claim 7, the bin fetch unit to fetch the data corresponding to the vertex indicated by the plurality of pointers from a frame buffer.

10. The apparatus of claim 7, the bin fetch unit to fetch the data corresponding to the vertex indicated by the plurality of pointers from a main memory device.

11. The apparatus of claim 7, further including a vertex cache, the bin fetch unit to fetch the data corresponding to the vertex indicated by the plurality of pointers from the vertex cache.

12. The apparatus of claim 11, wherein the vertex cache includes a plurality of entries, each entry to store 32 bytes of vertex data.

13. A system, comprising:
a processor;
a memory controller coupled to the processor;
a main memory coupled to the memory controller; and
a graphics controller including a bin fetch unit to fetch primitive data from a plurality of bin storage areas located in the main memory, the primitive data including a plurality of pointers to indicate a memory location for data corresponding to a vertex, the bin fetch unit further to fetch the data corresponding to the vertex indicated by the plurality of pointers from an additional bin storage area.

14. The system of claim 13, the bin fetch unit to fetch the data corresponding to the vertex indicated by the plurality of pointers from a frame buffer coupled to the graphics controller.

15. The system of claim 13, the bin fetch unit to fetch the data corresponding to the vertex indicated by the plurality of pointers from the main memory.

16. The system of claim 13, the graphics controller further including a vertex cache, the bin fetch unit to fetch the data corresponding to the vertex indicated by the plurality of pointers from the vertex cache.

17. The system of claim 16, wherein the vertex cache includes a plurality of entries, each entry to store 32 bytes of vertex data.