US 20050017970A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2005/0017970 A1

Howson (43) Pub. Date: Jan. 27, 2005

(54) THREE-DIMENSIONAL COMPUTER GRAPHICS SYSTEM

(76) Inventor: **John William Howson**, St. Albans (GB)

Correspondence Address:
FLYNN THIEL BOUTELL & TANIS, P.C.
2026 RAMBLING ROAD
KALAMAZOO, MI 49008-1699 (US)

(21) Appl. No.: **10/795,561**

(22) Filed: **Mar. 5, 2004**

(30) **Foreign Application Priority Data**

Jul. 25, 2003 (GB) 0317479.4

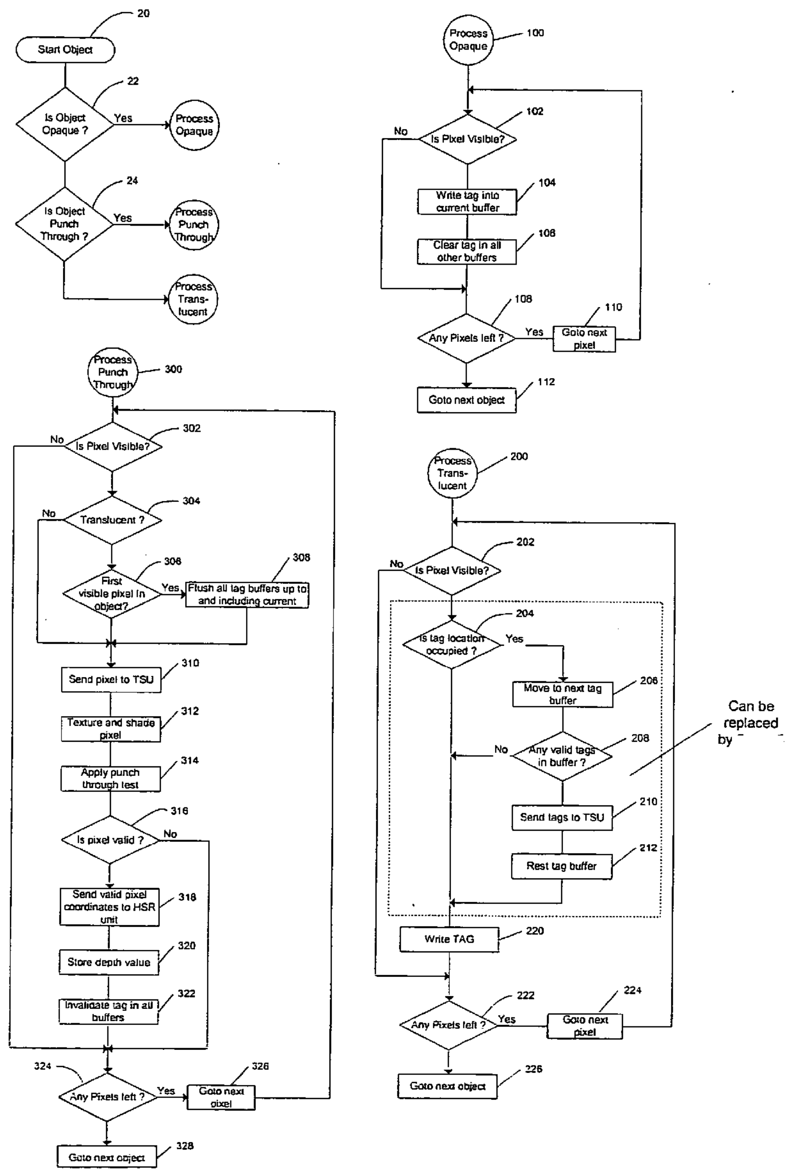
Publication Classification

(51) **Int. Cl.**^{7} **G06T 15/40**

(52) **U.S. Cl.** **345/421**

(57) **ABSTRACT**

A method and apparatus provided for rendering three-dimensional computer graphic images which include both translucent and opaque objects. A list of objects which may be visible in the image is determined and for each pixel in the image a determination is made as to whether or not an object for the list may be visible at that pixel. A data tag is stored for a transparent object determined to be visible at a pixel and the tag and object data are passed to a texturing and shading unit when a translucent object is determined to be overwriting the location in the tag buffer already occupied by another data tag.



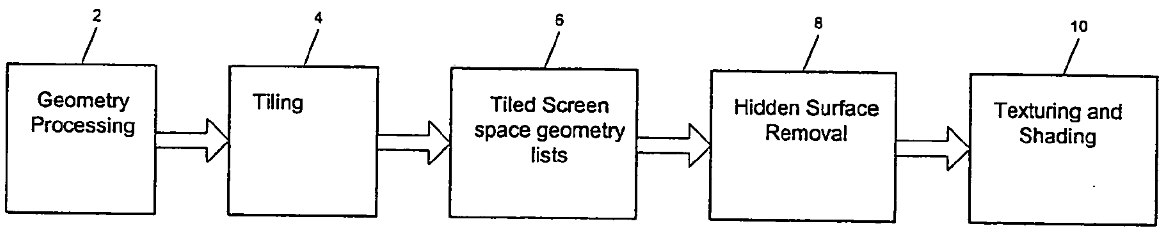


Figure 1

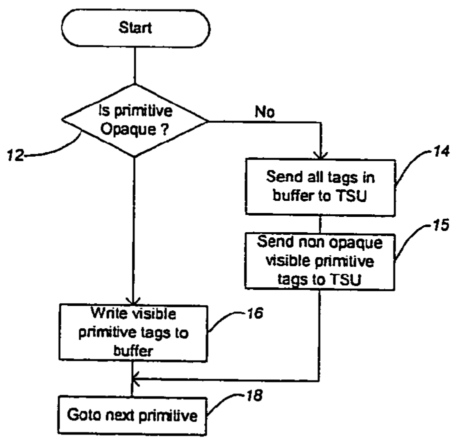

Start
Is primitive Opaque ?
12
No
Send all tags in buffer to TSU
14
Send non opaque visible primitive tags to TSU
15
Write visible primitive tags to buffer
16
Goto next primitive
18

Figure 2

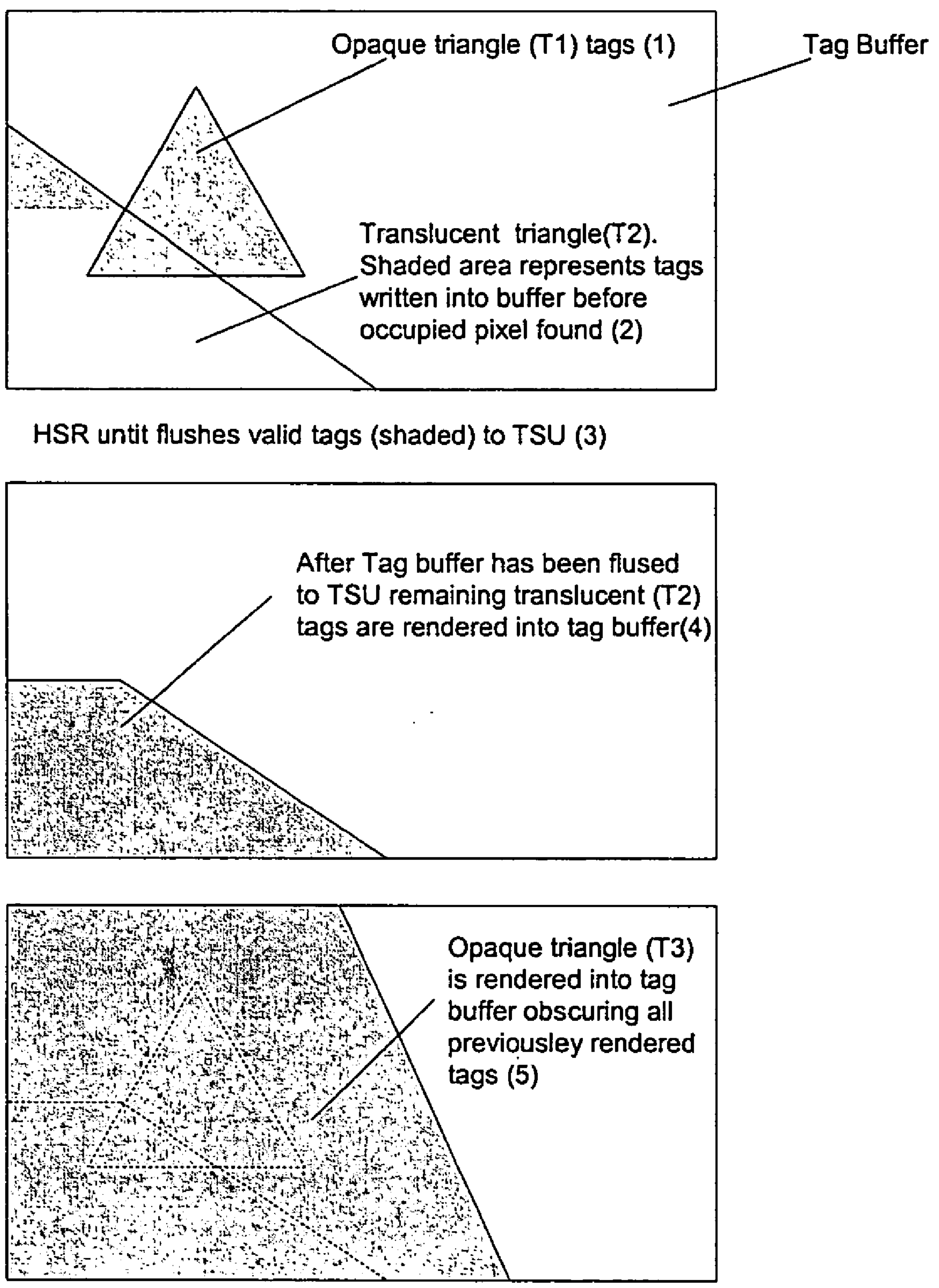

Opaque triangle (T1) tags (1)
Tag Buffer
Translucent triangle(T2). Shaded area represents tags written into buffer before occupied pixel found (2)
HSR untit flushes valid tags (shaded) to TSU (3)
After Tag buffer has been flused to TSU remaining translucent (T2) tags are rendered into tag buffer(4)
Opaque triangle (T3) is rendered into tag buffer obscuring all previousley rendered tags (5)

Figure 3

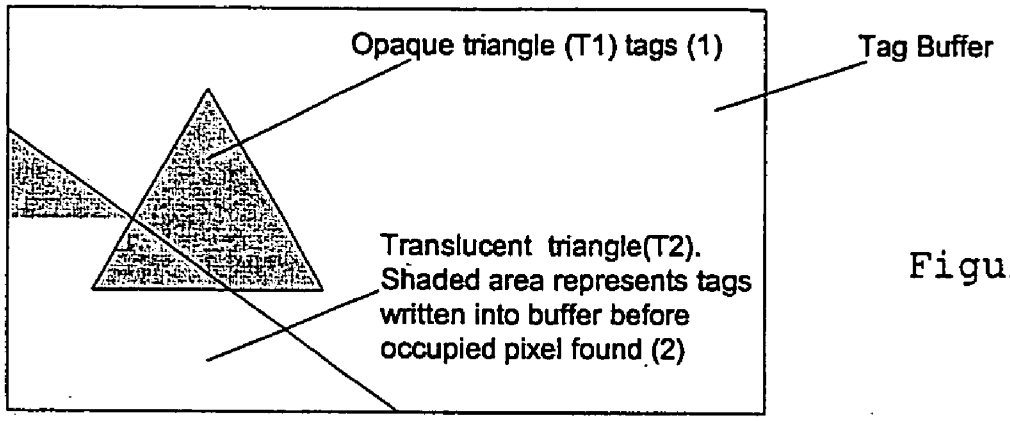

Opaque triangle (T1) tags (1)
Tag Buffer
Translucent triangle(T2). Shaded area represents tags written into buffer before occupied pixel found (2)

Figure 3A

HSR unit flushes valid tags (shaded) to TSU (3)

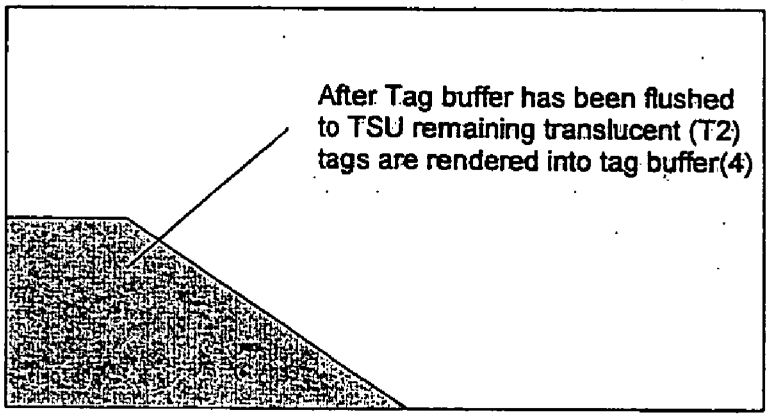

After Tag buffer has been flushed to TSU remaining translucent (T2) tags are rendered into tag buffer(4)

Figure 3B

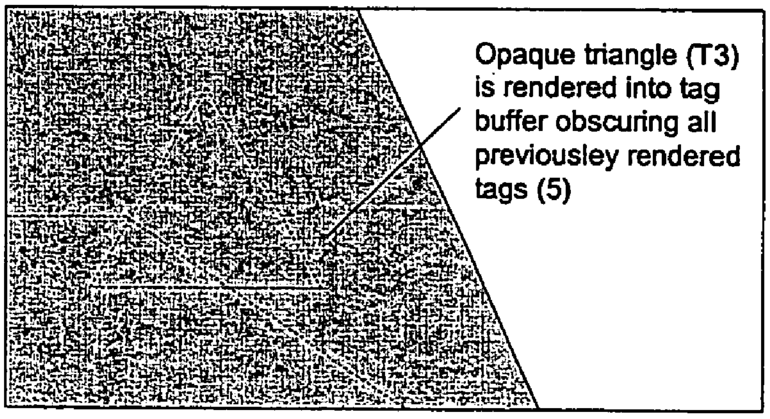

Opaque triangle (T3) is rendered into tag buffer obscuring all previousley rendered tags (5)

Figure 3C

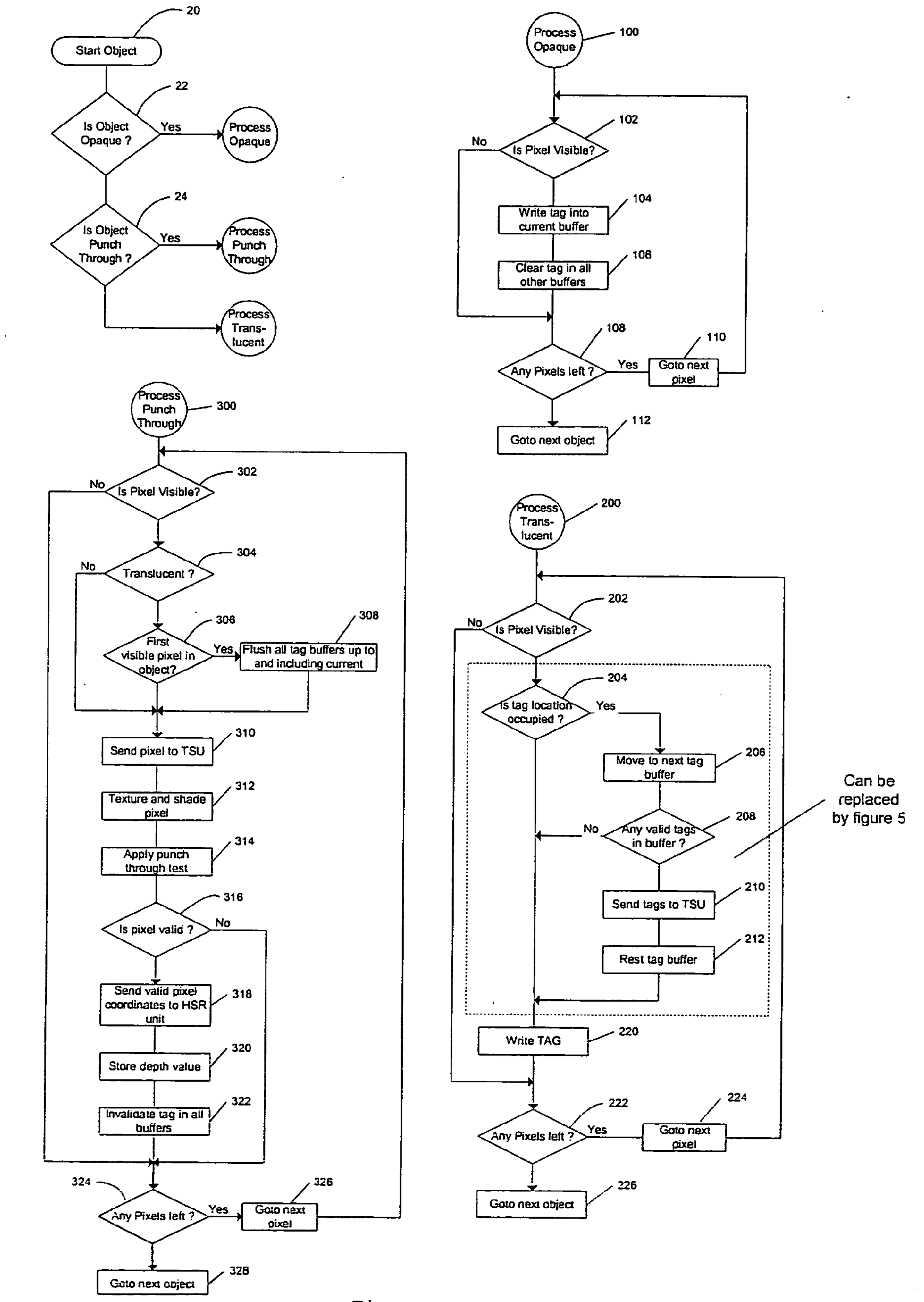

20
Start Object
22
Is Object Opaque ?
Yes
Process Opaque
24
Is Object Punch Through ?
Yes
Process Punch Through
Process Trans-lucent
Process Opaque
100
102
No
Is Pixel Visible?
104
Write tag into current buffer
106
Clear tag in all other buffers
108
110
Any Pixels left ?
Yes
Goto next pixel
112
Goto next object
Process Punch Through
300
302
No
Is Pixel Visible?
304
No
Translucent ?
306
308
First visible pixel in object?
Yes
Flush all tag buffers up to and including current
310
Send pixel to TSU
312
Texture and shade pixel
314
Apply punch through test
316
Is pixel valid ?
No
Send valid pixel coordinates to HSR unit
318
320
Store depth value
322
Invalidate tag in all buffers
324
326
Any Pixels left ?
Yes
Goto next pixel
328
Goto next object
Process Trans-lucent
200
202
No
Is Pixel Visible?
204
Is tag location occupied ?
Yes
Move to next tag buffer
206
208
No
Any valid tags in buffer ?
210
Send tags to TSU
212
Rest tag buffer
Can be replaced by figure 5
Write TAG
220
222
224
Any Pixels left ?
Yes
Goto next pixel
226
Goto next object

Figure 4

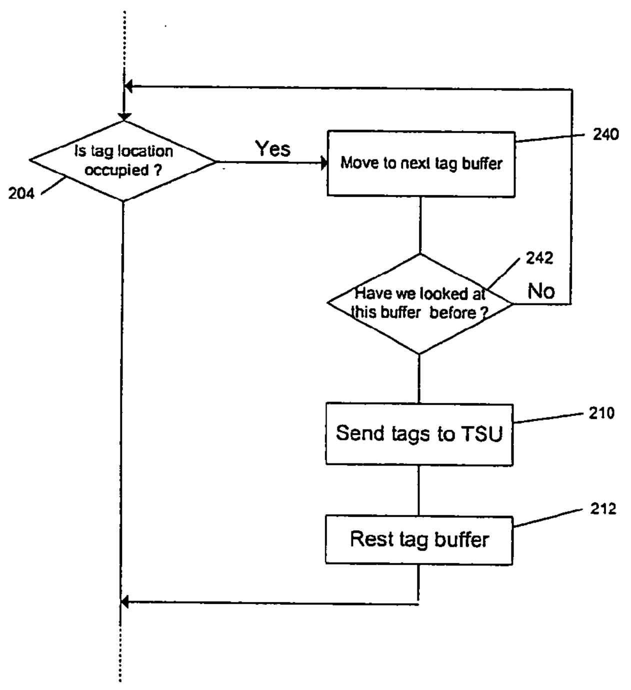


Figure 5

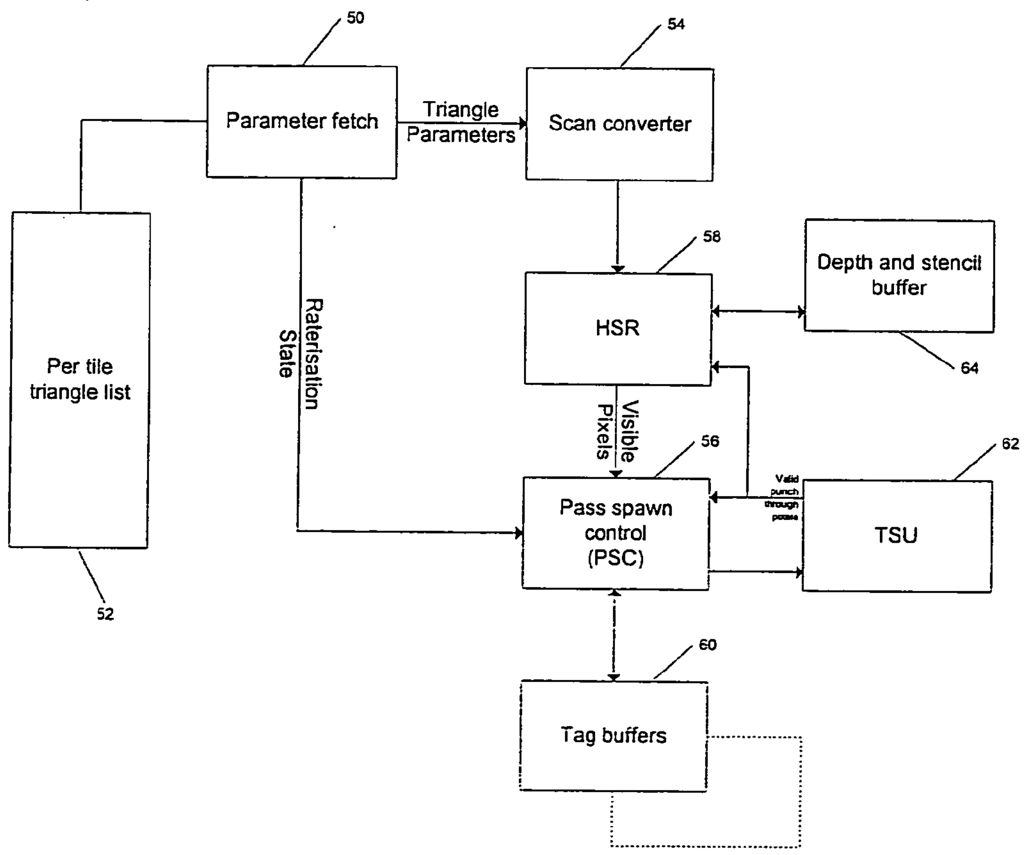


Figure 6

THREE-DIMENSIONAL COMPUTER GRAPHICS SYSTEM

[**0001**] This invention relates to a 3-dimensional computer graphics system and in particular to methods and apparatus which reduce the number of times the data assigned to each pixel have to be modified when rendering an image in such a system.

[**0002**] Tile based rendering systems are known. These break down an image to be rendered into a plurality of rectangular blocks or tiles. The way in which this is done and the subsequent texturing and shading performed is shown schematically in **FIG. 1**. This shows a geometry processing unit **2** which receives the image data from an application and transforms it into screen space using a well-known method. The data is then supplied to a tiling unit **4**, which inserts the screen space geometry into object lists for a set of defined rectangular regions, or tiles, **6**. Each list contains primitives (surfaces) that exist wholly or partially in a sub-region of a screen (i.e. a tile). A list exists for every tile on the screen, although it should be borne in mind that some lists may have no data in them.

[**0003**] Data then passes tile by tile to a hidden surface removal unit **8** (HSR) and from there to a texturing and shading unit **10** (TSU). The HSR unit processes each primitive in the tile and passes to the TSU only data about visible pixels.

[**0004**] Many images comprise both opaque and translucent objects. In order to correctly render such an image, the HSR unit must pass "layers" of pixels which need to be shaded to the TSU. This is because more than one object will contribute to the image data to be applied to a particular pixel. For example the view from the inside of a building looking through a pane of dirty glass requires both the geometry visible through the glass, and then the pane of glass itself to be passed to the TSU. This process is referred to as "pass spawning".

[**0005**] Typically, a tile based rendering device of the type shown in **FIG. 1** will use a buffer to hold a tag for the front most object for each pixel in the tile currently being processed. A pass is typically spawned whenever the HSR unit **8** processes a translucent object, before the visibility test is performed. This results in all currently visible tags stored in the buffer followed by the visible pixels of the translucent object being sent to the TSU, i.e. more than one set of pixel data being passed for each pixel.

[**0006**] The flow diagram of **FIG. 2** illustrates this approach. In this, a determination is made at **12** as to whether or not a primitive being processed is opaque. If it is not, then the buffer of tags is sent at **14** to the TSU **10**. All visible tags for the non-opaque primitives are then also passed to the TSU at **15**, the HSR unit **8** will then move onto the next primitive at **18**. If the primitive is determined to be opaque at step **12** then its tags are written into the buffer at **16** before moving onto the next primitive at **18**. The tag is a piece of data indicating which object is visible at a pixel. More than one tag per pixel is required when translucent objects cover opaque objects.

[**0007**] Use of the approach above means that opaque pixels that are not covered by translucent pixels, and are potentially obscured by further opaque objects may be passed to the TSU unnecessarily. In addition to this, a translucent object is passed to the TSU even if an opaque object subsequently obscures it.

SUMMARY OF THE INVENTION

[**0008**] The presence of the tag buffer in the above description enables modifications to be made to the pass spawning rules (also described above) that allow the removal of some unnecessary passes.

[**0009**] In an embodiment of the present invention, rather than spawning a pass at the point a translucent object is seen, the translucent tags are rendered into the tag buffer in the same manner as opaque objects and a pass only spawned at the point a visible translucent pixel is required to be written to a location that is already occupied. Further, as the translucent object tags are now being rendered into the tag buffer there is no need to pass them immediately to the TSU. Therefore, in the event of them being subsequently obscured they may be discarded.

[**0010**] The invention is defined with more precision in the appended claims to which reference should now be made.

BRIEF DESCRIPTION OF THE DRAWINGS

[**0011**] Preferred embodiments of the invention will now be described in detail by way of example with reference to the accompanying drawings in which:

[**0012**] **FIG. 1** shows a block diagram of a tile based rendering system discussed above;

[**0013**] **FIG. 2** shows a flow chart of a known pass spawning system;

[**0014**] **FIG. 3** (*a*)-(*c*) shows a sequence of three triangles being rendered using modified pass spawning rules embodying the invention;

[**0015**] **FIG. 4** is a flow diagram of the embodiment of the invention;

[**0016**] **FIG. 5** is an enhancement to **FIG. 4**; and

[**0017**] **FIG. 6** is a block diagram of an embodiment of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

[**0018**] In **FIG. 3 a** sequence of three triangles is shown being rendered using a set of modified pass spawning rules. In **FIG. 1***a*, an opaque triangle T1 is rendered into a tag buffer. If a translucent triangle T2 is rendered on top of the visible opaque pixels then the HSR unit must pass those pixels to the TSU before it can continue to rasterise the translucent tags. The opaque triangle is encountered in scan line order as T2 is rasterised. Thus all the previously rasterised pixels of T1 and the pixels of T2 are passed to the TSU. This leaves the remainder of the translucent object as shown in **FIG. 3***b*. An opaque triangle T3 is then rendered into the tag buffer as shown in **FIG. 3***c*. This triangle T3 obscures all of T1 and T2.

[**0019**] It will be seen that tags from T1 and T2 have been passed unnecessarily to the TSU in spite of the improved rules. The triangles passed unnecessarily are shown by dotted lines in **FIG. 3***c*. In addition to this, if T3 had been translucent and was subsequently obscured by another object, all tags from **T2** would be passed to the TSU.

[**0020**] If more than one tag buffer is provided then this can further reduce the number of unnecessary passes. In a system with N tag buffers a minimum of N obscured passes can be removed. This is achieved by switching to a new tag buffer each time an attempt is made to write a translucent pixel to an occupied location, until all tag buffers have been written to. At this point, the first tag buffer is passed to the TSU.

[**0021**] If an opaque pixel is written to the current tag buffer this will result in the same pixel being invalidated in all other buffers, thus removing any obscured pixels. This can be done as any pixel that is written to by an opaque primitive will only be composed of the data generated by that primitive. The flow chart of **FIG. 4** illustrates how the pass spawning rules behave in this case.

[**0022**] After the start at **20** of processing an object, as in **FIG. 2**, a determination is made as to the type of the object at **22** and **24**. If an object is determined to be opaque at **22** then the system executes the opaque processing path at **100**, if not and the object is determined to be punch through at **24** then punch through processing path is executed at **300** otherwise translucent processing is executed at **200**.

[**0023**] If the object is opaque it is processed from **100** on a pixel by pixel basis. For each pixel within the object the system first determines its visibility at **102**. If a pixel is not visible then the system skips to **108** to determine if there any more pixels left in the object. If a pixel is visible then its tag is written into the current tag buffer at **104**, the tags in all other buffers are cleared at **106**. The system then determines at **108** if any more pixels are left to process from the current object, if there are it moves to the next pixel at **110** and continues processing from **102**. If there are no more pixels to process in the object then the system moves to the next object at **112** and returns to **20**.

[**0024**] 3D computer graphics often use what is termed as "punch through" objects. These objects use a back end test to determine if a pixel should be drawn. For example, before a pixel is written to the frame buffer its alpha value can be compared against a reference using one of several compare modes, if the result of this comparison is true then the pixel is determined to be visible, if false then it is not. Pixels that are determined to not be visible do not update the depth buffer. It should be noted that this test can be applied to both opaque and partially translucent objects. This technique is common in 3D game applications because it allows complex scenes such as forests to be modelled using relatively few polygons and because a traditional Z buffer can correctly render punch through translucency irrespective of the order in which polygons are presented to the system.

[**0025**] As the update of the depth buffer is dependant on the results of the back end test tile based rendering (TBR) systems must handle these objects in a special manner. Existing TBR systems will first spawn a pass as if the punch through objects were transparent, the punch through object is then passed to the TSU which then feeds back visibility information to the HSR unit which updates the depth buffer as necessary.

[**0026**] The handling of punch through objects can be optimised. If punch through is applied to an opaque object it will either be fully visible or not visible at all, this allows them to be treated as opaque with respect to the flushing of the tag buffers. Specifically at the point an opaque punch through object is received any pixels that are determined to be visible by the hidden surface removal (HSR) unit are passed directly to the TSU. The TSU then feeds back pixels that are valid to the HSR unit which will, for the fed back pixels, update the depth buffer as appropriate and invalidate the same pixels in the tag buffer. The later is possible as the act of passing the punch through pixels to the TSU means that they have already been drawn so any valid tags at the same locations in the tag buffer are no longer needed. If multiple tags buffers are present then the tags are invalidated across all buffers.

[**0027**] Partially transparent punch through data requires all tag buffers up to and including the current tag buffer to be flushed to the TSU. This is because the transparency may need to be blended with any overlapped tags currently contained in the tag buffer. Alternatively, the punch through tags may be passed to the TSU with their states modified such that they will not update the frame buffer image and their tags are written to the next buffer as dictated by the punch through test. This allows objects that lie under the punch through object that are subsequently obscured not to be rendered. However, this is at the cost of potentially rendering the punch through object twice, once to determine pixel visibility and once to render the final image if it is not obscured. The impact of rendering the punch through object twice could be reduced by splitting the TSU pixel's shading state into that required to determine punch through state and that required to render the final image.

[**0028**] As far as the flow diagram of **FIG. 4** is concerned, where the object is determined to be a punch through at **24**, the system jumps to Process Punch Through **300**. The object is then processed pixel by pixel, first determining visibility at **302**. If a pixel is not visible the system skips to **324**. If a pixel is visible a further test is made at **304** to determine if the object pixel is also transparent, and if this is determined to be the first visible pixel within the object at **306** then all tag buffers up to and including the current buffer are flushed at **308**, i.e. sent to the TSU. The test for translucency is performed per pixel so that the tag buffers do not get flushed in the event of the object not being visible. The pixels for the object itself are then sent to the TSU at **310** where texturing and shading are applied at **312** using well-known methods. A punch through test is then applied at **314** and the validity of the pixel determined at **316**. If the pixel is found to be invalid at **316**, e.g. it fails the alpha test, the system skips to **324**. If the pixel is valid its coordinates are passed back to the HSR unit at **318**, which will then store the pixels depth value to the depth buffer at **320** and invalidate the corresponding tag in all tag buffers at **322**. The system then determines if there are any more pixels to be processed in the object at **324**, if there are it moves to the next pixel at **326** and jumps back to **302** to continue processing. If no more pixels are present in the object the system moves to the next object and returns to **20**.

[**0029**] When the punch through determination at **24** is negative then the object must be translucent and the system jumps to Process Translucent **200**. The object is then processed pixel by pixel, first determining visibility at **202**. If a pixel is not visible the system skips to **222**. If the pixel is visible the system determines if location in the current tag buffer is occupied at **204**. If the current tag buffer location is determined to be occupied the system will move to the next tag buffer at **206**. A determination is then made as to whether or not there are any valid tags in the buffer at **208** and if there are, they are sent to the TSU at **210** and the tag buffer reset at **212**. If there are no valid tags then a tag is written at **220** and the system goes on to the next pixel or object as described for opaque objects.

[**0030**] As opaque objects invalidate tags across all buffers, the pass spawning rules can be further extended such that passes only spawn when no tag buffer can be found into which a translucent pixel can be written. **FIG. 5** illustrates these updated rules which can be used to replace the portion of the flow diagram of **FIG. 4** surrounded by a dotted line. Instead of the determination at **208** as to whether there are any valid tags in the buffer, a determination is made as to whether or not this buffer has been looked at before. If it has, then the flow moves onto **210** and **212**. If it has not, flow passes back to **204** where a determination is made as to whether or not the tag location is occupied. If it is, then the diagram moves to the next tag buffer at **240** before again determining whether or not that buffer has been looked at at **242**.

[**0031**] A further enhancement can be made to single and multiple buffer implementations. Rather than flushing the whole tag buffer at the point that no unoccupied pixel can be found for a translucent object, only those tags that would be overwritten by the translucent pixel are flushed to the TSU. The main disadvantage of this approach is that it can result in the partial submission of an object to a TSU which can result in it being submitted many times. This leads to additional state fetch and set up costs in the TSU. This could be alleviated by submitting all pixels with the same tag value to the TSU rather than only those that are overlapped by the translucent object. Alternatively, the tag buffer could be subdivided into square/rectangular sections such that when the above condition occurs only the section of the tag buffer containing the conflict would be flushed. This approach also potentially results in multiple submissions of tags to the TSU but to a lesser extent.

[**0032**] A block diagram of a preferred embodiment of the invention is shown in **FIG. 6**. This comprises a parameter fetch unit **50** which reads in per tile lists of triangles and rasterisation state from a per triangle list **52**. These are then passed to a scan converter **54** and to a pass spawn control unit **56** respectively. The scan converter **54** generates position, depth and stencil values for each pixel within each triangle and passes them to an HSR unit **58**. The HSR unit determines the visibility of each pixel and passes this information onto the pass spawning control unit (PSC) **56**. This unit has access to two or more tag buffers **60** which are cycled through in a circular manner. For opaque pixels the PSC unit writes the tag of the triangle to the current tag buffer and invalidates the corresponding location in the other buffer. For a translucent pixel the PSC unit checks to see if the location in the current tag buffer is valid. If it is, then it switches to the other tag buffer. If a tag location is not valid it writes the translucent tag and moves onto the next pixel. If the tag location is valid then the tag buffer is flushed to the texturing and shading unit **62**. At this point all locations in the tag buffer are marked as invalid and the translucent tag is written to the buffer. For an opaque punch through pixel the PSC passes all visible pixels directly to the TSU **62**. The TSU determines pixel validity as appropriate and returns the status of those pixels to the HSR and PSC units **58**,**56**. The HSR and PSC units then update depth buffer and invalidate locations in tag buffers respectively for valid pixels. Translucent punch through pixels behave the same as opaque punch through except that the PSC unit will flush all currently valid tags in the tag buffers to the TSU before proceeding. This process is repeated for all pixels in all triangles within the tile.

[**0033**] When the parameter fetch unit **50** determines that there are no more triangles in the current tile it signals to the pass spawning unit **56** to flush any remaining valid tags from the tag buffers to the TSU **62**. The parameter fetch unit then proceeds to read the parameter list for the next tile and repeats the process until all tiles that make up the final image have been rendered. It should be noted that all of the units, with the exception of the parameter fetch, can be modified to operate on multiple pixels in parallel, thereby speeding up the process.

[**0034**] The HSR unit **58** has access to a depth and stencil buffer **64** in which the depth and stencil values for each pixel within each triangle are stored.

1. A method for rendering 3-dimensional computer graphic images which include translucent and opaque objects comprising the steps of:

determining a list of objects which may be visible in the image;

for each pixel in the image determining whether or not an object from the list may be visible at that pixel;

storing a data tag for transparent object determined to be visible pixel in a tag buffer; and

passing tag data and object data to a texturing and shading unit (TSU) when a translucent object is determined to be overwriting a location in the tag buffer already occupied by another data tag.

2. A method according to claim 1 including the steps of:

storing at least one additional tag each time a translucent object is determined to be visible at a pixel the additional tag being stored in a separate location to any previously stored tag; and

in which the step of passing tag and object data to the TSU is performed from the first stored tag when all tag storage locations are full.

3. A method according to claim 2 including the step of:

clearing all tags when an opaque surface is determined to be visible at a pixel; and

storing a tag associated with the opaque surface.

4. A method according to claim 1 including the step of initially subdividing a scene to be rendered into a plurality of rectangular sub-regions and subsequently performing the method for each sub-region.

5. A method according to claim 1 including the step of determining whether or not an object is a punch through object and performing a different process when a punch through object is detected.

6. A method according to claim 5 including the step of determining whether the punch through object is an opaque object, and if an opaque object is detected, clearing all tags

from the tag buffer for pixels intersected by that object and sending the punch through object data to the TSU.

7. A method according to claim 5 including the step of determining whether the punch through object is translucent and if a translucent object is detected passing all data tags and object data for a pixel to the TSU, including the object data for the translucent punch through object.

8. An apparatus for rendering 3-dimensional computer graphic images which include translucent and opaque objects comprising:

means for determining a list of objects which may be visible in the image;

means for determining for each pixel in the image whether or not an object from the list may be visible at that pixel;

a tag buffer for storing a data tag for a translucent object determined to be visible at a pixel; and

means for passing tag data and object data to a texturing and shading unit (TSU) when a translucent object is determined to be overwriting a location in the tag buffer already occupied by another data tag.

9. An apparatus according to claim 8 including means for storing at least one additional tag each time a translucent object is determined to be visible at a pixel, the pixel tag being stored in a separate location in the tag buffer to any previously stored tag; and

in which the means for passing tag and object data to the TSU performs this step from the first stored tag when all tag storage locations are full.

10. An apparatus according to claim 9 including means for clearing all tags when an opaque surface is determined to be visible at a pixel; and

means for storing in the tag buffer a tag associated with the opaque surface.

11. An apparatus according to claim 8 including means for initially sub-dividing a scene to be rendered into a plurality of rectangular sub-regions and subsequently operating on each sub-region.

12. An apparatus according to claim 8 including means for determining whether or not an object is a punch through object and means for performing a different process when a punch through object is detected.

13. An apparatus according to claim 12 including means for determining whether or not a punch through object is an opaque object and means for clearing all tags from the tag buffer for pixels intercepted by an opaque punch through object and means for sending the opaque punch through object data to the TSU.

14. An apparatus according to claim 12 including the steps of determining whether or not the punch through object is a translucent and if a translucent punch through object is detected passing all data tags and object data for a pixel to the TSU, including the object data for the translucent punch through object.

* * * * *