# Efficient State Management for Tile-Based 3D Graphics Architectures

Iosif Antochi, Ben Juurlink, Stamatis Vassiliadis
Computer Engineering Laboratory, EEMCS
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
Phone: +31 15 2783644 Fax: +31 15 2784898
E-mail: {tkg|benj|stamatis}@ce.et.tudelft.nl

Petri Liuha
NOKIA Research Center
Tampere, Finland
E-mail: petri.liuha@nokia.com

*Abstract*— **Tile-based rendering is a promising technique for low-power, 3D graphics platforms. This technique decomposes a scene into smaller regions called tiles and renders the tiles one-by-one. The advantage of this scheme is that a small memory integrated on the graphics accelerator can be used to store the color components and z (depth) values of one tile, so that accesses to these values are local, on-chip accesses which consume significantly less power than off-chip accesses. Tile-based rendering, however, requires that the primitives (commonly triangles) and state changes are sorted into bins corresponding to the tiles. In this paper we determine the optimal state change operations (e.g., enable/disable z testing, create/delete a texture) that should be included for each tile. Experimental results obtained using several suitable 3D graphics workloads show that various trade-offs can be made and that, usually, better performance can be obtained by trading it for memory.**

*Keywords*— **3D graphics architecture, tile-based rendering, state management.**

## I. Introduction

Low-power mobile devices are becoming increasingly popular. They are no longer used only for mobile telephony or as a PDA, but also for a broader spectrum of applications like e-commerce and interactive 3D games [1]. However, the resources offered by such platforms are rather limited.

Tile-based rendering (also called chunk rendering or bucket rendering) is a technique in which the scene is divided into tiles and the tiles are rendered independently. Tile-based rendering appears to be promising for low-power implementation because the color components and depth values of the primitives of a certain tile can be stored in small, on-chip buffers and only the pixels visible in the final scene need to be written to the external framebuffer. Since external memory accesses dissipate significant amounts of power, replacing them by accesses to local buffers reduces the power consumption.

Tile-based rendering requires, however, that the primitives are sent to the accelerator in tile-based order. In other words, they need to be sorted into bins or buckets that correspond to the tiles. Moreover, the initial state information required to render each primitive should also be distributed among primitives.

Determining which state change operations can be safely removed and when is not trivial. For instance, if a delete texture command is encountered while rendering the current tile, the texture can be safely deleted only when all primitives (from all tiles) that use this texture are rendered or it can be deleted when multiple copies of the texture are kept in memory. Also, including all the state change operations to each tile is not a practical solution since it requires duplicating large amounts of state variables (e.g., texture objects) for each tile.

In this paper we analyze the state change data sent to a tile-based renderer and propose algorithms to optimize it.

This paper is organized as follows. After describing the previous work on state management for tile-based renderers in Section II, we describe in more detail the OpenGL state information in Section III. In Section IV are described the state management algorithms we considered for our implementation. Experimental results are presented in Section V, and the conclusions are given in Section VI.

## II. Related work

Tile-based architectures were initially proposed for high-performance, parallel renderers [2], [3], [4]. Since tiles cover non-overlapping parts of a scene, the triangles that intersect with these tiles can be rendered in parallel. In such architectures it is important to balance the load on the parallel renderers [5]. These studies are, however, not very related to our study since we consider a low-power architecture in which the tiles are rendered sequentially one-by-one.

Tile-based rendering has also been used in power-aware architectures [6], [7].

Two specific issues related to tile-based rendering are

```
EnableDepth
Triangle(1)
DisableDepth
Triangle(2)
EnableDepth
Triangle(3)
```

Fig. 1. Static state - initial instruction stream fragment

per tile primitive sorting and state management.

While the process of determining which primitives belong to a tile was already discussed in the literature [8], there are less details about state management for tile-based state management algorithms.

## III. OpenGL State Information

The OpenGL state information can be divided into two parts. The first part is the static state information, that is the state information that needs to be stored irrelevant of the application. For instance, the information that describes the state of the depth unit is always defined. The second part of the state information is the dynamic state information. The dynamic state information contains the state information which is application dependent. For instance, the texture state depends on the number of textures loaded by the application.

### *A. Static State Information*

The static state information part of the OpenGL state machine is usually less than the dynamic state information and it has no side effects. More precisely, duplicating the static information to each tile does not affect the execution semantics of OpenGL. However, since the primitives that do not overlap a tile are deleted from the instruction list of the respective tile, some state information might be also eliminated. Figure 1 depicts a fragment of instructions sent to the rasterizer. Assuming that triangle 1 overlaps tile 2, triangle 2 overlaps tile 1, and triangle 3 overlaps tiles 1 and 2, the instruction stream that might be generated by a tile-based driver is depicted in Figure 2. The emphasized state changing instructions can be also deleted from the tiled instruction stream. By eliminating the unnecessary state change instructions, the data traffic to the rasterizer is decreased. Nevertheless, we note that determining if a state instruction can be eliminated might consume actually more system bandwidth than sending it directly to the rasterizer.

**Tile1**

```
   EnableDepth
   DisableDepth
   Triangle(2)
   EnableDepth
   Triangle(3)
```

**Tile2**

```
   EnableDepth
   Triangle(1)
   DisableDepth
   EnableDepth
   Triangle(3)
```

Fig. 2. Static state - tiled instruction stream fragment

### *B. Dynamic State Information*

In this section we describe in more detail the state information required to be stored for texturing. Figure 3 depicts the texture state information organization. Each texture unit supported by the hardware has a link to a current texture object. Each texture object, identified by a texture id, contains information such as: the texture image format, the width and the height of the largest texture level, the minification and magnification functions, and links to texture images for valid texture levels. A texture object can be bound to a texture unit using the BindTexture command. When a texture object is no longer needed, it can be deleted using the DeleteTexture command.

## IV. State Management Algorithms for Tile-Based Rendering

This section presents several algorithms that can be used to correctly handle the state information when using a tile-based rendering model.

### *A. Partial Rendering Algorithm*

In this algorithm, whenever an instruction that has side-effects is encountered (e.g., DeleteTexture) in the input stream, the driver renders all previously buffered instructions and then executes the instruction. While partial rendering is a solution to rendering commands having side effects, it might also introduce significant rendering overhead. For example, consider that the instruction stream depicted in Figure 4 was sent to the rasterizer. The assumptions are the same as described in Section III-A. The tile-based driver or a tile-based rasterizer state management engine could issue the instruction stream depicted in

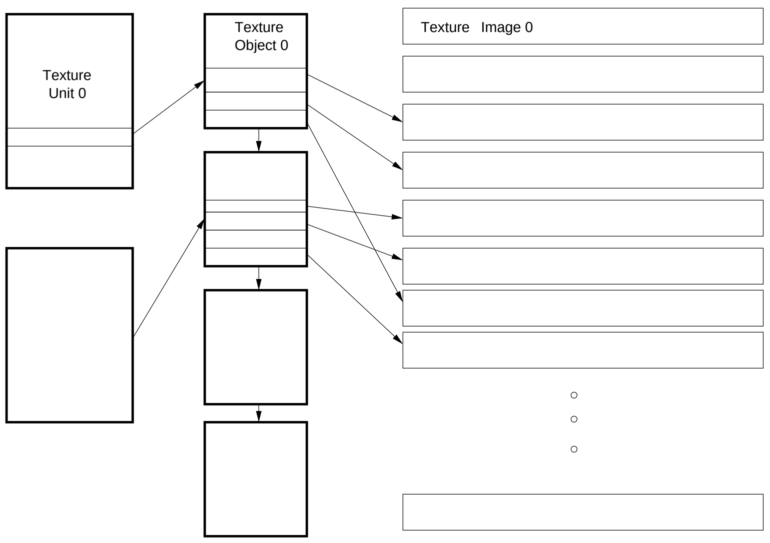

Fig. 3. Texture state information.

```
Start Frame
   CreateTexture(i)
   MakeCurrentTexture(i)
   Triangle(1)
   Triangle(2)
   DeleteTexture(i)
   CreateTexture(i)
   MakeCurrentTexture(i)
   Triangle(3)
End Frame
```

Fig. 4. Partial rendering - initial instruction stream

Figure 5. Since tiles are rendered sequentially, all the instructions preceding the DeleteTexture(*i*) instruction, in all tiles, must be rendered so that all the primitives using texture *i* are rendered and thus the DeleteTexture instruction can be executed. For each partial rendering the introduced overhead consists of saving and reloading the contents of each tile and also extra context (all the state information) save and reload operations.

### B. Delayed Execution Algorithm

In this algorithm, when commands that affect dynamic state, e.g., Delete Texture or TextureImage, are encountered in the input stream, the driver will postpone their execution until all the primitives depending on them are rendered or the end of the current frame is reached. For instance, assume that a DeleteTexture was encountered. As long as no request to create new textures are received, thus no reuse required, for the texture ids, the execution of the DeleteTexture command can be safely delayed until

```
Start Frame
Tile 1
   c1=SaveCurrentContext
   RestoreTileFromGlobalBuffer
   CreateTexture(i)
   MakeCurrentTexture(i)
   Triangle(2)
   SaveTileToGlobalBuffer
   c2=SaveCurrentContext
Tile 2
   RestoreContext(c1)
   RestoreTileFromGlobalBuffer
   MakeCurrentTexture(i)
   Triangle(1)
   SaveTileToGlobalBuffer
   DeleteTexture(i)
Tile1
   c1=SaveCurrentContext
   RestoreTileFromGlobalBuffer
   CreateTexture(i)
   MakeCurrentTexture(i)
   Triangle(3)
   SaveTileToGlobalBuffer
Tile 2
   RestoreContext(c1)
   RestoreTileFromGlobalBuffer
   MakeCurrentTexture(i)
   Triangle(3)
   SaveTileToGlobalBuffer
End Frame
```

Fig. 5. Tiled instruction stream using partial rendering

```
Start Frame
Tile 1
    c1=SaveCurrentContext
    RestoreTileContentsFromGlobalBuffer
    CreateTexture(i)
    MakeCurrentTexture(i)
    Triangle(2)
    MarkDeleteTexture(i)
    RenameTexture(i,j)
    MakeCurrentTexture(j)
    Triangle(3)
    SaveTileContentsToGlobalBuffer
Tile 2
    RestoreContext(c1)
    RestoreTileContentsFromGlobalBuffer
    MakeCurrentTexture(i)
    MakeCurrentTexture(i)
    Triangle(1)
    MakeCurrentTexture(j)
    Triangle(3)
    SaveTileContentsToGlobalBuffer
After Last Tile
    DeleteTexture(i)
    MoveTextureLinks(i,j)
End Frame
```

Fig. 6. Delayed tiled instruction stream using delayed commit

all the primitives that use the texture are executed. However, if the application requests a new a texture id from the OpenGL front-end, and obtains a texture id that was deleted on the same frame but not committed, then the tile-based driver must create a new texture object that will be linked with a new id, while the old texture object will remain accessible until all primitives using the old texture are rendered or until the end of the frame. Executing (committing) the commands when all primitives depending upon them are finished has a higher computational power than executing it at the end of the frame since it requires determining the last tile and the last primitive depending on it.

## V. Experimental Results

In order to compare the efficiency of the proposed algorithms we used the benchmarking suite proposed in [9]. It consists of 7 components: Q3L, Q3H, Tux, Aw, ANL, GRAZ, and DINO. The Q3L profile corresponds to a low resolution (320x240) demo of the Quake III 3D FPS game. The Q3H profile is based on the same demo as Q3L only that it uses higher resolution (640x480). Tux is a 3D racing game (guide a penguin) available on Linux platforms. The Aw (Awadvs-04) profile is part of the Viewperf 6.1.2 package. The NAT, GRAZ, and DINO are 3D VRML models for which "fly-by" scenes were created and traced. The traces were fed to our modified Mesa library. The Mesa library performed primitive backface culling and generated lists of remaining primitives. The list of primitives were sent to our tile-based accelerator simulator, where different primitive to tile bounding box tests were used. We used a tile size of 32x16 pixels, and the window sizes were 320x240 for Q3L, and 640x480 for the other benchmark suite components.

Figure 7 depicts the percentage of the state information and primitives sent to the accelerator. The average percentage of unoptimized tile-based state information across the benchmarks is 44%. This high percentage is obtained due to the overhead of state information replication and also additional tile-based specific state change information such as load and save tile operations. The Aw component has the lowest percentage of state change due to the fact that most of the state change is performed if the first frames and there is little state variation from frame to frame. GRAZ, Tux, and Q3, on the other hand, combine multiple texture and blending modes and depth tests so they require more state information to be sent to the accelerator.

Figure 8 presents the state changed sent to the rasterizer using direct transfer or lazy commit (delayed) methods. The delayed method reduces the number of writes to the accelerator, by up to 58%, by filtering the state information and eliminating unnecessary writes, thus providing the optimal state traffic with the expense of a small additional memory. The obtained results show that the state information for the Q3H, GRAZ, and ANL components can be significantly reduced. The state traffic for the Aw component, however, could not be decreased since there were no unnecessary state changes.

## VI. Conclusions

In this paper we have presented several state management algorithms for tile-based renderers. While in traditional (non tile-based) rendering the state information traffic can be negligible compared to the traffic generated by the primitives, in tile-based rendering architectures, since the state information might need being duplicated in multiple streams, the required processing power and generated traffic can increase significantly. Moreover, removing primitives from the instruction stream of a tile depends only on the primitive position and the tile coordinate. To remove a state change instruction from the instruction stream of a tile, information about the previous or the following state change instructions and/or primitives is required. Thus, in order to send an optimal state change stream to the accelerator, i.e., use minimal bandwidth, ad-

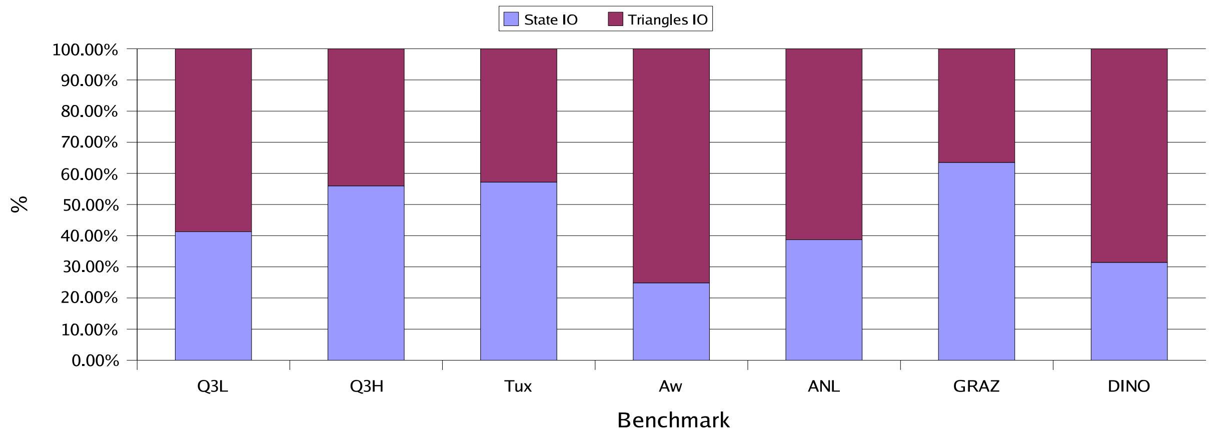

Fig. 7. Percentage of state information and triangles sent to the accelerator per frame.

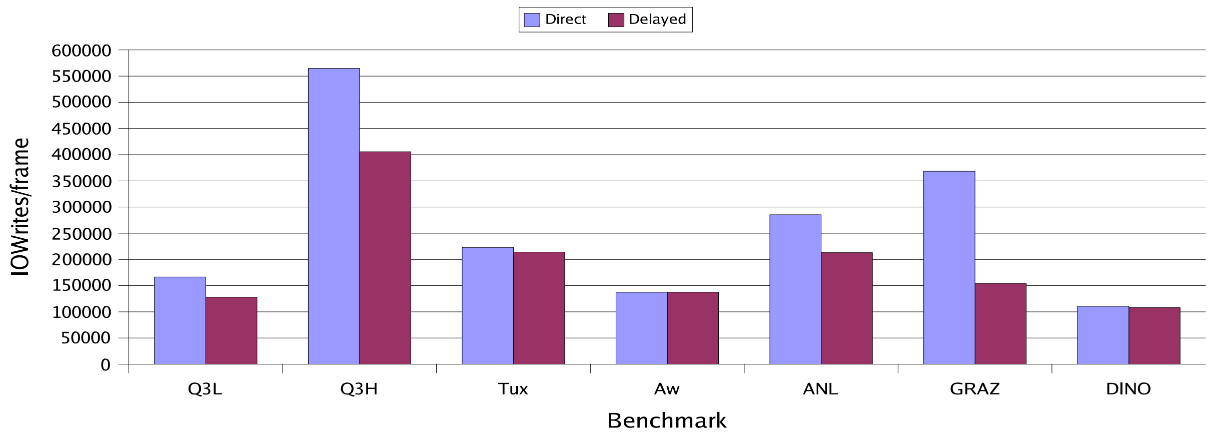

Fig. 8. Average number of state information writes to the accelerator per frame.

ditional processing power and more processor bandwidth is required. By sending an optimal state change stream to the accelerator, the state change traffic to the accelerator was decreased up to 58%.

## REFERENCES

[1] The New York Times. Cell Phone Games Take Leap Into 3D. Available at `http://news.com.com/Cell+phone+games+take+leap+into+3D/2100-1043_3-5237341.html`.

[2] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, B. Tebbs G. Turk, and L. Israel. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. *Computer Graphics, Vol. 23, No. 3, pp. 79–88*, July 1989.

[3] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A Sorting Classifi cation of Parallel Rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, 1994. IEEE Computer Society Press.

[4] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A Stream Processing Framework for Interactive Rendering on Clusters. In *Proc. 29th Annual Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH 2002)*, pages 693–702, 2002.

[5] Carl Mueller. The Sort-First Rendering Architecture for High-Performance Graphics. In *Proc. 1995 Symp. on Interactive 3D Graphics*, pages 75–84. ACM Press, 1995.

[6] PowerVR. 3D Graphical Processing (Tile Based Rendering - The Future of 3D), White Paper. http://www.beyond3d.com/reviews/videologic/vivid/PowerVR_WhitePaper.pdf, 2000.

[7] Emile Hsieh, Vladimir Pentkovski, and Thomas Piazza. ZR: A 3D API Transparent Technology for Chunk Rendering. In *Proc. 34th ACM/IEEE Int. Symp. on Microarchitecture (MICRO-34)*, 2001.

[8] Iosif Antochi, Ben Juurlink, Stamatis Vassiliadis, and Petri Liuha. Scene Management Models and Overlap Tests for Tile-Based Rendering. In *Proc. EUROMICRO Symp. on Digital System Design*, pages 424–431, 2004.

[9] Iosif Antochi, Ben Juurlink, Stamatis Vassiliadis, and Petri Liuha. GraalBench: A 3D Graphics Benchmark Suite for Mobile Phones. In *Proc. ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)*, pages 1–9, June 2004.