Efficient Tile-Aware Bounding-Box Overlap Test for Tile-Based Rendering

I. Antochi, B. Juurlink, S. Vassiliadis
Computer Engineering Laboratory
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: {tkg, benj, stamatis}@ce.et.tudelft.nl

P. Liuha
Nokia Research Center
Visiokatu-1, SF-33720
Tampere, Finland
E-mail: petri.liuha@nokia.com

Abstract

Tile-based rendering appears to be a promising technique for low-cost, low-power 3D graphics platforms. This technique decomposes a scene into tiles and renders the tiles independently. It requires, however, that the primitives are sorted into bins that correspond to the tiles, which can be very time-consuming and may require a lot of memory bandwidth. The most often used test to determine if a primitive and a tile overlap is the bounding box test. This test checks if the 2D axis aligned bounding box of the primitive overlaps the tile and comprises four comparisons in the worst case. In this paper we show that the efficiency of the bounding box test can be improved significantly by adaptively varying the order in which the comparisons are performed depending on the position of the current tile. Experimental results obtained using several 3D graphics workloads show that the dynamic bounding box test reduces the average number of comparisons per primitive by 26% on average compared to the best performing static version in which the order of the comparisons is fixed.

1. Introduction

With the advent of the system-on-chip (SOC) design paradigm for embedded systems, 3D graphics accelerators for mobile platforms, in particular mobile phones, are becoming increasingly popular. Since mobile phones are powered by batteries, a 3D graphics accelerator for such devices should dissipate as little energy as possible.

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. The test commonly employed to determine if a primitive (triangle in most cases) overlaps with a tile is the so-called bounding box (BBOX) test. It consists of four comparisons that test if the maximum (resp. minimum) x- or y-coordinate of the triangle is smaller (resp. larger) than the minimum (resp. maximum) x- or y-coordinate of the tile. If one of these tests succeeds then the triangle definitely does not intersect with the tile. If all tests fail then the triangle might intersect with the tile.

Sorting the primitives can be performed by the accelerator or by the host processor. Because the amount of chip area of our low-cost target system (a 3D graphics accelerator for mobile phones) is severely limited, we assume it needs to be performed by the host CPU. If sorting is performed in software then the order in which the comparisons are applied can have a significant impact on the performance. For example, for the tile in the upper-left corner we expect that most triangles are located to the east and/or south of it.

In this paper we present several versions of the bounding box test that dynamically vary the order of the comparisons depending on the position of the tile that is currently being rendered. It is shown that these dynamic versions of the bounding box test perform significantly better than the static version. Because the primitives have to be sorted at frame rate, it is important that the sorting process to be executed as fast as possible.

The paper is organized as follows. Related work is briefly described in Section 2. Section 3 describes the bounding box test in more detail and presents our dynamic versions. Experimental results are provided in Section 4, and concluding remarks are given in Section 5.

0-7803-8558-6/04/$20.00

2. Related Work

Tile-based architectures were initially proposed for high-performance, parallel renderers [5, 8, 7]. 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 [9]. 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 [10, 6]. However, no details have been provided on how the primitives are sorted into bins corresponding to the tiles. Furthermore, in the literature mentioning the usage of bounding box tests [10, 6, 4, 3] there is no mention on how the bounding box tests are implemented or performed. Some implementation details were provided in [2] where several scene management algorithms for tile-based architectures were presented, based on a two step model.

3. Static and Dynamic Versions of the Bounding Box Test

In this section we describe the BBOX test in detail and present our dynamic versions of the BBOX test.

3.1. Overview of the BBOX Test

Fig. 1 illustrates the BBOX test. It consists of two steps. First, the 2D axis aligned BBOX of the primitive is computed. Thereafter, it is determined if the BBOX intersects the tile. Let a triangle *Tr* be defined by three points p_1, p_2, and p_3, whose x coordinates are given by $p_i.x$ and whose y-coordinates are given by $p_i.y$. Then the BBOX of *Tr* is defined by the tuple (*BBOX.MinX*, *BBOX.MinY*, *BBOX.MaxX*, *BBOX.MaxY*) where

$$\begin{aligned} \mathit{BBOX.MinX} &= \text{MIN}(p_1.x, p_2.x, p_3.x) \\ \mathit{BBOX.MinY} &= \text{MIN}(p_1.y, p_2.y, p_3.y) \\ \mathit{BBOX.MaxX} &= \text{MAX}(p_1.x, p_2.x, p_3.x) \\ \mathit{BBOX.MaxY} &= \text{MAX}(p_1.y, p_2.y, p_3.y). \end{aligned}$$

Let the tile be given by the tuple (*T.MinX*, *T.MinY*, *T.MaxX*, *T.MaxY*). Then a possible implementation of the BBOX test in C is:

```
if (BBOX.MaxX < T.MinX)    /* Test 1 */
   return NoOverlap;

if (BBOX.MinX >= T.MaxX)    /* Test 2 */
   return NoOverlap;
```

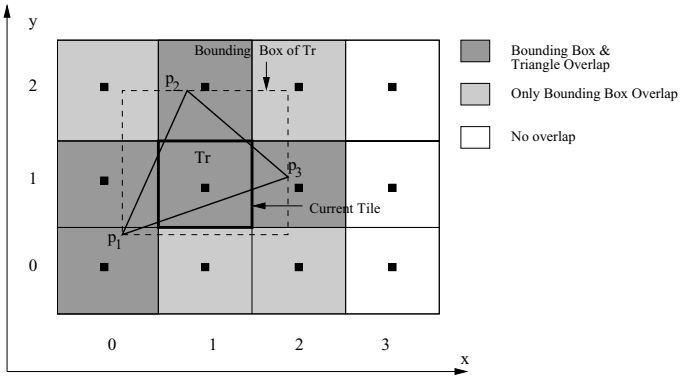

Figure 1. Triangle to tile BBOX test

```
if (BBOX.MaxY < T.MinY)    /* Test 3 */
   return NoOverlap;

if (BBOX.MinY >= T.MaxY)    /* Test 4 */
   return NoOverlap;

return MightOverlap;
```

We remark here that if all tests fail then the triangle might overlap the tile but it might also be the case that the BBOX intersects with the tile but the triangle does not. This is illustrated in Fig. 1. The light-grey tiles intersect with the BBOX but not with the triangle. Thus the BBOX test performs only a partial classification. There are two ways to deal with this situation. Either an additional, exact but more expensive test is performed, or the triangle is sent to the rasterizer in which case no fragments are generated for the triangle if it does not overlap the tile. In practice the BBOX test is preferred to exact but more expensive tests because of its simplicity and because it is quite accurate.

3.2. Primitive Sorting

Computing the BBOX of a triangle is in general more expensive than testing if the BBOX and the tile intersect. However, in some algorithms for sorting the primitives the BBOX calculation is performed only once per primitive while the second step of the BBOX test that checks if the BBOX intersects with the current tile is performed for every combination of primitive and tiles. The second step of the BBOX test, therefore, has a larger impact on the time consumption of the sorting scheme than the first step.

In particular, in [2] we have proposed and analyzed several algorithms for sorting the primitives into bins that correspond to the tiles. In this paper we focus on the TWO_STEP algorithm which has reasonable performance

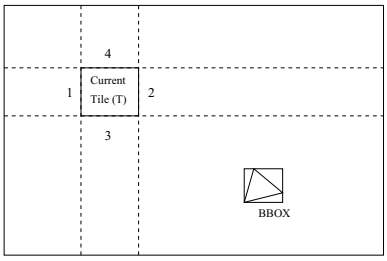

Figure 2. Triangle to tile BBOX test using 1D prediction with correlation

and does not require a significant amount of memory in addition to the scene buffer. This algorithm consists of two steps. In the first, the BBOX of each primitive is computed and stored. In the second step, the BBOXs of all primitives are scanned for each tile and if the BBOX test indicates a possible overlap, the primitive is sent to the rasterizer.

3.3 Static Bounding Box

The four comparisons required to determine if a BBOX and a tile overlap can be performed in an arbitrary order. This gives a total of 24 possible arrangements. However, not every order produces the same number of comparisons on average. In this section we discuss two versions of the static bounding box test that might generate a different number of comparisons on average.

A tile divides the screen into five, possibly intersecting regions: the tile itself, the region to the east of the tile ($x >= T.MaxX$), the region to the west of the tile ($x < T.MinX$), the region to the north ($y >= T.MaxY$), and the region to the south ($y < T.MinY$). If a certain test (comparison) fails, then there is a high probability that the test in the opposite direction along the same dimension succeeds. This is because after these two tests there is only a small region left where the BBOX of a primitive can be situated.

Fig. 2 illustrates the case in which first the region to the west of the tile is checked, then the opposite region along the same dimension (east), then the region to the south, and, finally, the region to the north. After performing only two comparisons (the horizontal intersection tests), all primitives that are completely located to the east or west of the tile are rejected. This usually leaves only a small number of triangles that require more than two comparisons. Of course, different orders are also possible (for example, north, south, west, east). However, assuming that the primitives are equally distributed over the scene, they should produce the same number of comparisons on average. We refer to this scheme as *STATIC1*.

To determine if *STATIC1* reduces the average number of comparisons, we will compare it to a scheme in which the first and second (and, hence, the third and fourth) comparison check different dimensions. For example, one possible order is west, south, east, north. Statistically, there should be no difference between the possible orders. We refer to this static variant as *STATIC2*.

3.4 Dynamic Bounding Box

As stated before, a tile divides the scene into four regions (five if we include the tile itself). The probability that a primitive is completely located in the largest region is the highest. This observation is the basis of our "dynamic" versions of the bounding box test.

We describe two dynamic schemes. In the first, referred to as *DYNAMIC1*, we first check the largest region. Thereafter, the opposite direction along the same dimension is tested. The third test examines the largest region in the other dimension, and the fourth test checks the remaining region. The second dynamic version of the bounding box test is referred to as *DYNAMIC2*. In this scheme, the comparison corresponding to the largest region is applied first, then the comparison corresponding to the second largest region, etc. The region to the east of the tile is the largest and checked first, then the region to the south, then the one to the north, and, finally, the region to the west.

We remark that although these schemes are called dynamic, the order in which the comparisons are applied depends only on the tile position and can be determined statically off-line. For example, for all tiles in the upper left sub-scene under the main diagonal, the order is east, south, north, west.

4. Experimental Results

In order to compare the efficiency of the proposed algorithms we used the benchmarking suite proposed in [1]. It consists of 7 components: Q3L, Q3H, Tux, Aw, NAT, 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

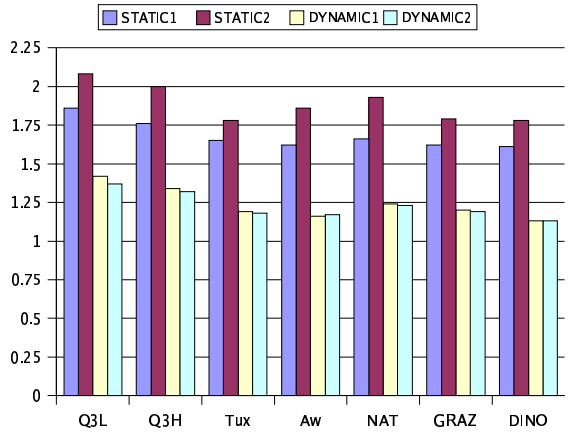

Figure 3. The average number of comparisons per primitive for each workload

32x16 pixels, and the window sizes were 320x240 for Q3L, and 640x480 for the other benchmark suite components.

Fig. 3 depicts, for each workload, the average number of comparisons per primitive required by each version of the bounding box test. As expected, the *STATIC1* scheme indeed performs better than the *STATIC2* scheme. On average, across the benchmarks, *STATIC1* requires 11% fewer comparisons than *STATIC2*. It can also be seen that there is little difference between the two dynamic versions. This can be explained as follows. If the largest region has been checked, then a large part of the second largest region has also been checked. Suppose, for example, that the largest region is to the east of the tile and that the second largest region is to the south. If the region to the east has been checked, then the region to the southeast of the tile has also been checked, while the region to the west has not. Nevertheless, we observe that *DYNAMIC2* performs slightly better than *DYNAMIC1*. Furthermore, both dynamic schemes require fewer comparisons than the best static version. On average, the best dynamic version *DYNAMIC2* requires 26% fewer comparisons than the best static scheme *STATIC1*.

The results above show that by dynamically varying the order in which the comparisons are performed depending on the position of the current tile indeed reduces the average number of comparisons needed to determine that a triangle does not intersect the tile.

5. Concluding Remarks

In this paper we have described several possible software implementations of the bounding box test. The static versions always perform the comparisons involved in the same order, while the dynamic versions base the order on the position of the current tile. The experimental results show that the dynamic scheme in which the comparison corresponding to the largest region is applied first, then the comparison corresponding to the second largest region, etc., requires the least comparisons on average (26% fewer comparisons than the best performing static version).

References

[1] I. Antochi, B. Juurlink, S. Vassiliadis, and P. 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) (to appear)*, June 2004.

[2] I. Antochi, B. Juurlink, S. Vassiliadis, and P. Liuha. Scene Management Models and Overlap Tests for Tile-Based Rendering. In *Proc. EUROMICRO Symp. on Digital System Design (DSD 2004) (to appear)*, 2004.

[3] M. Chen, G. Stoll, H. Igehy, K. Proudfoot, and P. Hanrahan. Simple Models of the Impact of Overlap in Bucket Rendering. In *Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 105–112, Lisbon, Portugal, 1998. ACM Press.

[4] M. Cox and N. Bhandari. Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC. In *Proc. 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 25–34. ACM Press, 1997.

[5] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, B. T. 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.

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

[7] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. 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.

[8] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A Sorting Classifi cation of Parallel Rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, 1994. IEEE Computer Society Press.

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

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