Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC

Michael Cox
MRJ/NASA Ames Research Center
Microcomputer Research Labs, Intel Corporation

Narendra Bhandari
Microcomputer Research Labs, Intel Corporation

Abstract

Bucket rendering is a technique whereby a scene is sorted into screen-space tiles and each tile is rendered independently in turn. We expect hardware-accelerated bucket rendering to become available on the PC, and in this paper we explore the effect of such accelerators on main memory bandwidth, bus bandwidth to the accelerator, and on increased triangle set-up requirements. The most important impact is due to the fact that in general primitives *overlap* multiple buckets, which is a direct cause of overhead. In this paper we evaluate bucket rendering that uses the most common algorithm for *bucket sorting*, one based on screen-aligned primitive *bounding boxes*. We extend previous techniques for analytically evaluating bounding box overlap of buckets, and together with experimental results use these to evaluate accelerators that may support 32x32 pixel tiles, and those that may support 128x128 pixel tiles. We expect the former to be possible with dense SRAM, the latter to be possible with DRAM embedded in a logic process (*embedded DRAM*). Our results suggest that embedded DRAM implementations can support bucket rendering *with bounding box bucket sorting*, but that SRAM implementations will likely be at risk with respect to overall system performance when bounding box bucket sorting is employed. These results suggest the requirement for more precise but *still* low-overhead bucket sorting algorithms when bucket rendering hardware is constrained to 32 x 32 tiles.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture – raster display devices; I.3.3 [Computer Graphics]: Picture/Image Generation – display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – visible surface algorithms.

Additional Key Words and Phrases: bucket rendering, chunk rendering, tile rendering, overlap.

1 Introduction

Many of the features that users want require significant memory bandwidth and/or memory capacity. At the high-end of the market for example [1], oversampling for geometry anti-aliasing is typically implemented in "brute-force" fashion with large replicated interleaved memories. On the other hand, the PC market constrains the growth of both the memory bandwidth and memory capacity available with graphics acceleration hardware. Package costs constrain the pin bandwidth to external memory, cost constrains external memory capacity, and die area constrains the capacity of on-chip memory.

1997 SIGGRAPH/Eurographics Workshop

Bucket rendering is an approach that can be employed when there is insufficient memory to hold the intermediate results of rendering. With this approach, the screen is partitioned into tiles, and the primitives of the scene are sorted into buckets that correspond to the screen tiles. Then, in turn, each bucket is rendered into a tile, and the tiles are collected to form a final image.

Bucket rendering has been around at least since Pixar's first implementation of RenderMan [12], and has been implemented in hardware by PixelPlanes 5 [5], PixelFlow [7] and by Apple in a shipping product [8]. It has more recently resurfaced as a central technique in Microsoft's Talisman architecture (where a "bucket" is referred to as a "chunk") [11].

The advantage of bucket rendering is that it enables algorithms to store only a tile of intermediate results when the algorithms would otherwise require a full screen of intermediate results. In particular, bucket rendering can enable such features as full-screen oversampling or A-buffering, with significantly less memory, or z-buffering without the requirement to store an external full z-buffer. Recent and anticipated advances in fabrication technology also make bucket rendering of interest now. Fairly dense on-chip SRAM has become commercially viable, and the major fabrication houses are actively pursuing processes to embed DRAM in a logic process (*embedded DRAM*). These advances potentially enable on-chip tiles for greater memory bandwidth, and potentially for decreased external memory capacity requirements.

We expect accelerators that employ/support bucket rendering will be designed for the PC platform. The features that become possible with fast on-chip memory and the potential reduction in external memory costs are compelling. Since the PC market is horizontal rather than vertical we should expect that bucket rendering accelerators will be available from more than one Independent Hardware Vendor (IHV), and that these vendors may make different hardware choices. Furthermore, software must be written (or modified) to use such accelerators and we should expect multiple Independent Software Vendors (ISVs) to make different software choices as well.

A critical choice that the IHV must make is the bucket size. An important decision that must be made by the ISV is the bucket sorting algorithm. Together, these have potentially significant impact on the PC architecture. In this paper we examine two bucket sizes in particular that the IHV may implement, one corresponding to an implementation using dense SRAM, the other to an implementation using embedded DRAM. We also examine the interaction of bucket size with the most common bucket sorting algorithm, bounding box bucket sorting.

This paper is organized as follows. First, we present an overview of the PC architecture that we assume. Next, we

	Scene							
	14	*15*	*20*	*22*	*25*	*39*	*41*	*42*
A average	192.2	121.5	3162.0	2989.0	1125.6	1590.9	139.2	52.7
A max	370.5	710.7	52788.0	74198.8	72256.9	20144.8	12909.2	35587.5
r average	2.3	4.2	10.0	28.4	10.9	34.8	10.9	2.4
r max	34.0	44.5	630.5	5212.1	525.0	857.2	154.9	33.0
Frac > 1K	0.00	0.00	0.48	0.35	0.17	0.33	0.03	0.00
N tris.	844	645	1760	677	2683	346	5209	6085
Resolution	440 x 402	440 x 402	632 x 484	632 x 484	632 x 484	632 x 434	632 x 434	632 x 434

Table 1. Scene statistics. A = bounding box area. r = aspect ratio. *Frac > 1K* = fraction of bounding boxes larger than 1K in area. N = number of triangles in the scene. *Resolution* = resolution at which the scene was traced.

discuss bucket sorting and the provenance of the most common algorithm. Following this, we discuss the issue of primitive overlap, and our experimental methodology in evaluating it. Then we extend previous analytical results to calculate overlap, and finally use these and gathered statistics to evaluate the effect of 32 x 32 and 128 x 128 tiles on the PC architecture when the software algorithm employs bounding box bucket sorting.

2 Architecture

We assume the PC architecture shown in Figure 1. In this architecture, the CPU (or *host*) communicates with both main memory and the Graphics Accelerator (GA) via Core Logic (CL). The path from the CL to the GA is via the Accelerated Graphics Port (AGP) which provides up to 512 Mbytes/sec of bandwidth dedicated to graphics. Typically associated with the GA is external memory employed for frame buffer, z-buffer, and (depending on the accelerator architecture) texture memory.

We assume that the buckets into which primitives are sorted are stored in main memory, and that the processor maintains buckets. Putting the buckets in the external graphics memory would drive that memory to significantly larger size, and would meet resistance from many of the potential customers of a GA. In any event, placing the buckets in external graphics memory would not provide graceful failure for large scenes.

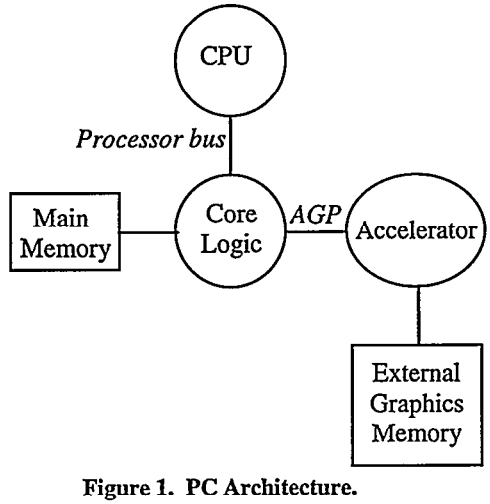


Figure 1. PC Architecture.

We assume that each bucket is written to the GA (or DMA'd) when the GA is ready to render that bucket. We assume on-chip memory in the GA for intermediate sample or fragment color values (and perhaps Z), and that when the GA is done rendering, it filters these intermediate results and writes final pixel values to the frame buffer (which is probably but need not be external graphics memory). After doing so, the GA is ready to render the next bucket.

3 Bucket Sorting

We assume that bucket sorting is done by the host, and that buckets are constructed and maintained in main memory by the application and/or graphics library. The issues in bucket sorting are non-trivial,[1] and we do not in this paper address more than the *type* of sorting that may be employed.

Probably the simplest algorithm that makes sense is *bounding box bucket sorting*. In this algorithm, the screen-aligned bounding box for each primitive is constructed (by *min(x), max(x), min(y), and max(y)*) and with simple comparisons each bounding box is placed in each bucket that the bounding box *overlaps*. This algorithm has the advantage that it is extremely cheap, easy to implement, and is robust. It has the disadvantage that it may place a bounding box in buckets that the inscribed primitive does not actually overlap. However, it has been employed in both software [2, 3, 12] and hardware [5, 7] renderers.

Alternatively, the application may calculate exactly the buckets that a primitive overlaps (*exact bucket sorting*). In general, exact sorting requires scan conversion of the primitive (a bucket is just a large pixel), and primitive set-up for such scan conversion done on the host is a duplication of the work done by triangle set-up on the accelerator. The advantage of exact bucket sorting is that it potentially reduces the number of buckets into which each primitive is placed. Its disadvantage is cost, complexity, and duplication of work. We are not aware of implementations that do exact bucket sorting, and we do not in this paper examine its cost or the extent to which it may reduce overlap.

There are other potential techniques intermediate between bounding box bucket sorting and exact bucket sorting in both cost and benefit. In particular, standard techniques such as trivial accept/reject culling (cf. [4]) may be modified for *bucket culling*. Primitives may be clipped to bucket boundaries, completely eliminating overlap (implicitly utilized in "patch renderers" such as [12]). Or primitives may be partitioned into

[1] Consider, for example, the semantics required to interleave 2D bitblt's with 3D primitives when the 3D primitives are oversampled. Consider also, for example, what is required to sort a triangle strip into buckets.

those sufficiently small that the cost of overlap is small, and those sufficiently large that their overlap cost is high; larger primitives may be clipped or exactly sorted. Except for the use of "splitting" in "patch renderers" such as RenderMan [12], we are unaware of implementations that employ these potential techniques.

In this paper, our point of view is from the hardware architecture. In particular, our concern is with the hardware implications of the *worst reasonable bucket sorting algorithm* that an ISV may employ. Thus, we consider in this paper the effect on the PC architecture when bucket sorting is by bounding box.

4 Overlap and Overhead

There is system overhead incurred by bucket rendering. When a primitive falls into multiple buckets (*overlaps* them), its processing requires additional

- main memory bandwidth,
- AGP bandwidth, and
- primitive set-up cycles from the accelerator.

If a primitive overlaps E_O buckets, it must be written E_O times to memory, read E_O times from memory across AGP, and set up for rasterization E_O times by the accelerator. Therefore overlap is a key statistic in the evaluation of the effect of bucket rendering on the components of the PC. Consequently, the *tile size* supported by the accelerator is a key metric: the smaller the tile the larger the overlap. Tile size is one area where the IHV's choices impact the potential performance of the overall system (involving memory and AGP bandwidth).

Below we describe the experimental methodology we have employed to examine expected overlap from bucket rendering, and the effect of tile size on overhead.

5 Experimental Methodology

We have instrumented internal research software at Intel in order to gather statistics from existing synthetic databases. The Intel Scene Management (ISM) software is a scene management and rendering software package [10]. We have instrumented ISM to trace upon command every polygon rendered during a "fly through". For each polygon, we calculate the bounding box, the bounding box aspect ratio, into which buckets the primitive would be placed by bounding box sorting, and we mark those buckets. During rendering, we perform the bucket calculation for bucket sizes 16x16, 32x32, 64x64, 128x128, and 256x256. Statistics are then written to a file and analyzed off-line. We have assumed that the accelerator has fixed bucket sizes, and that it can be configured to apply bucket rendering to arbitrary window resolutions. For resolutions that are not a multiple of the bucket size, we have assumed that the first bucket is window-aligned in both x and y, but that the last bucket in x or y may partially overlap the window (i.e. that not all pixels in the bucket may be valid).

We have traced and evaluated multiple frames from several databases, and have chosen the frames with the greatest polygon count as our samples. The test scenes for which results we include in this paper are:

Scene	Description
14	Sphere. Traced to generate statistics from curved surfaces at arbitrary orientation.
15	Teapot.
20	Scene of a house from the "Village" database, an advanced environment for PC rendering (courtesy of Evans and Sutherland).
22	Another scene from the "Village".
25	More intricate scene from the "Village".
39	Computer-generated terrain, courtesy of Turner Whitted and Numerical Design Limited (NDL). The NDL package that generated this supports Level-of-Detail (LOD). This scene generated with LOD enabled.
41	Computer generated terrain from NDL package, LOD disabled.
42	Animated character leaving a box – modeled with bezier patches and tessellated into polygons.

The statistics we have gathered from these scenes are shown in Table 1. We report the average and maximum bounding box area A, the average and maximum aspect ratio of bounding boxes r, as well as the number N of triangles, and the resolution at which the trace was made. Aspect ratio is calculated as follows: divide the longer side of the bounding box by the shorter. The fraction of primitives with bounding boxes larger than 1K pixels is discussed later. Pictures of these scenes can be found following the references at the end of this paper.

6 Analytical Tools to Calculate Overlap

Overlap has previously been considered primarily in the context of parallel rendering and parallel graphics hardware architectures (cf. [2, 3, 9]). Molnar reports (and credits to Eyles) an expected-case closed form for overlap when bounding box bucket sorting is employed [9]. However, the Molnar/Eyles equation is in terms of the width w and height h of the bounding box, and offers no substantial guidance concerning an appropriate relationship between the two (i.e. an expected *aspect ratio*). Some researchers have assumed a rectangular bounding box such that $w = rh$ for some r [3], and some graphics hardware architects have reluctantly assumed square bounding boxes for lack of better data. We begin with the Molnar/Eyles equation, briefly deriving it for completeness, and then consider aspect ratio. We require the following definitions:

Term	Definition
w	Bounding box width.
h	Bounding box height.
s	Bounding box side when $s = w = h$.
r	Bounding box aspect ratio (w/h when $w > h$, h/w when $h > w$).
A	Bounding box area, $A = w*h$.
S	Square tile side width/length.
$\lfloor f \rfloor$	Integer part of f (i.e. *floor(f)*).
$[f]$	Fractional part of f (i.e. $f - \lfloor f \rfloor$).

The Molnar/Eyles equation for expected-case overlap of square buckets is:

$$E_O = 1 + \frac{w}{S} + \frac{h}{S} + \frac{wh}{S^2} \quad (1)$$

It can be derived briefly as follows. First, note that a bounding box of size wh must cover at least $(\lfloor w/S \rfloor + 1)*(\lfloor h/S \rfloor + 1)$ buckets.[2] It may cover an extra column $(\lfloor w/S \rfloor + 2)*(\lfloor h/S \rfloor + 1)$ or row $(\lfloor w/S \rfloor + 1)*(\lfloor h/S \rfloor + 2)$, or an extra column and row $(\lfloor w/S \rfloor + 2)*(\lfloor h/S \rfloor + 2)$. These cases depend on the chance alignment of w and h with respect to the bucket. Let us define the random variables X and Y to be the number of buckets hit by the bounding box in screen-coordinate x and y, respectively, and also assume that these random variables are independently distributed. Then

Pr[w does not cover extra column]
$= Pr[X = \lfloor w/S \rfloor + 1]$
$= 1 - \{w/S\}$

Pr[w does cover extra column]
$= Pr[X = \lfloor w/S \rfloor + 2]$
$= \{w/S\}$

Similarly for the random variable Y. The standard compound probabilities can be computed by

Pr[no extra column, no extra row]
$= Pr[X = \lfloor w/S \rfloor + 1] * Pr[Y = \lfloor h/S \rfloor + 1])$
$= (1 - \{w/S\})*(1 - \{h/S\})$

Pr[extra column only]
$= Pr[X = \lfloor w/S \rfloor + 2] * Pr[Y = \lfloor h/S \rfloor + 1])$
$= \{w/S\}*(1 - \{h/S\})$

Pr[extra row only]
$= Pr[X = \lfloor w/S \rfloor + 1] * Pr[Y = \lfloor h/S \rfloor + 2])$
$= \{h/S\}*(1 - \{w/S\})$

Pr[extra column, extra row]
$= Pr[X = \lfloor w/S \rfloor + 2] * Pr[Y = \lfloor h/S \rfloor + 2])$
$= \{w/S\} * \{h/S\}$

We can then calculate expected overlap E_O from the standard calculation of the mean of independently distributed random variables

$E[XY] = E[X]*E[Y] =$
$\Sigma_j X_j * P[X_j] * \Sigma_k Y_k * P[Y_k]$

Following this derivation equation 1 can be verified.

6.1 Aspect ratio

Let us define the aspect ratio of a bounding box by

$r = w/h$ (when $w > h$)
$r = h/w$ (when $h \geq w$)

Consider the population of bounding boxes in a given scene with a given aspect ratio r and fixed area A. The sides of these bounding boxes are related to square bounding boxes of the same area A by some constant c where

$w = cs$
$h = (1/c)s$
$s = \sqrt{A}$.

Restricting $c \geq 1$ without loss of generality, we have that c and aspect ratio r are related by $r = c^2$. Substituting these observations in equation 1, expected overlap given r is $E_{O|r}$

$$E_{O|r} = 1 + \left(\sqrt{r} + \frac{1}{\sqrt{r}}\right) * \frac{s}{S} + \left(\frac{s}{S}\right)^2 \quad (2)$$

Now, to solve E_O in terms of $E_{O|r}$ over the full population of bounding boxes would require the distribution of r in closed form, or the expected values of $\sqrt{r}$ and $1/\sqrt{r}$. We do not pursue either of these directions in the current paper. However, we do note that this function is at a minimum when $r=1$ (recall that by definition $r \geq 1$).

This means that when the bounding box is square, we expect overlap to be at a minimum. As the bounding box becomes more of a rectangle, we expect overlap to increase. Of course, when the bounding box is substantially larger than a bucket, the third term dominates. *But this equation certainly suggests that a triangle smaller than a bucket will cover more buckets when its bounding box is long and thin than when its bounding box is square.*

6.2 Experimental results

Scatter plots of bounding box area vs. bounding box ratio are shown in Figures 2 through 9. First, an incidental observation deserves note. In most of these plots there is strong correlation of increasing area with increasing aspect ratio (Figure 8 shows this trend notably). We believe these "lines" to represent underlying triangles from the same object(s), that stretch due to perspective as they increase in area closer to the viewer. However, for the current paper we have not investigated further.

Second, as can be seen there are in general substantially many primitives with large aspect ratios that are smaller than a 32x32 bucket (i.e. smaller than 1K in size). *These plots combined with equation 2 should lead us to expect more overlap than predicted assuming square bounding boxes.*

Experimentally measured overlap is compared with predicted overlap in Table 2. We have calculated expected overlap E_O from equation 2, *assuming square bounding boxes (i.e. r=1). Contrary to our expectation, there is a close fit of expected to observed overlap by assuming that bounding boxes are square.*

[2] Technically it must cover $(\lfloor (w-\varepsilon)/S \rfloor + 1)*(\lfloor (h-\varepsilon)/S \rfloor + 1)$ buckets, but a derivation that begins this way is neither instructive nor tangibly more accurate.

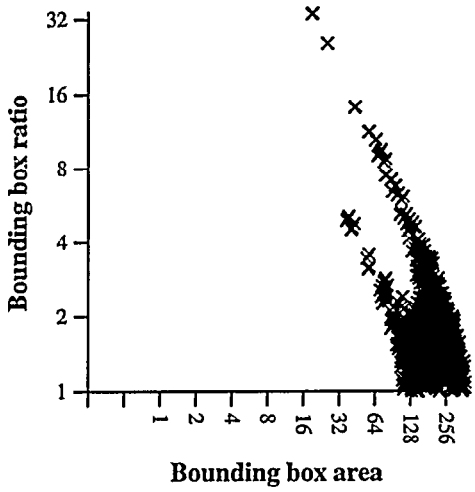


Figure 2. Scene 14.

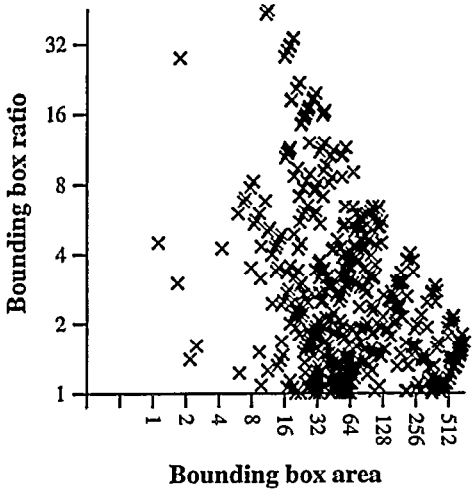


Figure 3. Scene 15.

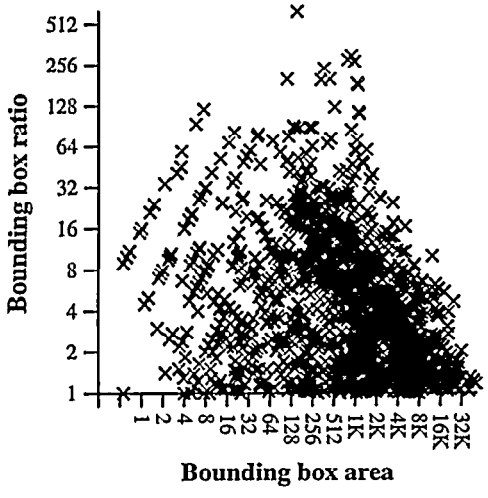


Figure 4. Scene 20.

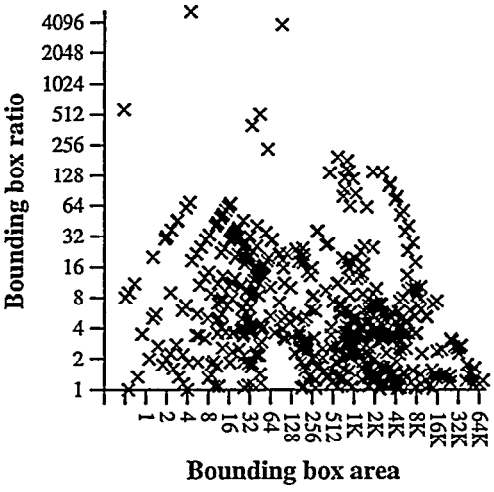


Figure 5. Scene 22.

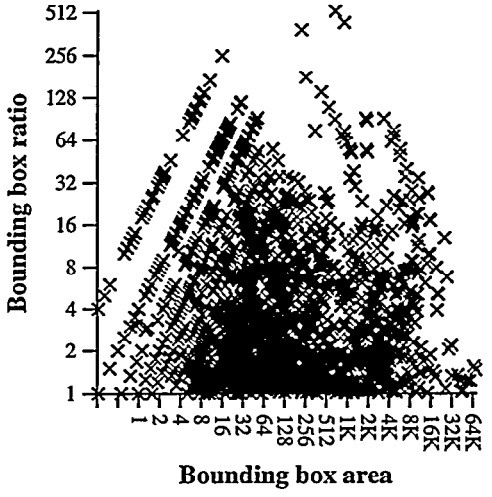


Figure 6. Scene 25.

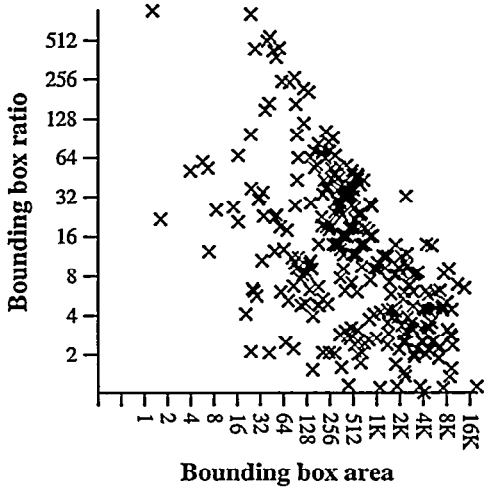


Figure 7. Scene 39.

	S=16		S=32		S=64		S=128		S=256	
Scene	*Obs.*	E_O	*Obs.*	E_O	*Obs.*	E_O	*Obs.*	E_O	*Obs.*	E_O
14	3.6	3.5	2.1	2.1	1.5	1.5	1.3	1.2	1.1	1.1
15	2.8	2.9	1.8	1.8	1.4	1.4	1.2	1.2	1.1	1.1
20	19.8	20.4	7.2	7.6	3.1	3.5	1.9	2.1	1.4	1.5
22	18.8	19.5	6.8	7.3	3.2	3.4	1.9	2.0	1.5	1.5
25	8.7	9.6	3.7	4.2	2.0	2.3	1.4	1.6	1.3	1.3
39	14.1	12.2	5.8	5.0	2.9	2.6	1.7	1.7	1.3	1.3
41	2.5	3.0	1.6	1.9	1.2	1.4	1.1	1.2	1.1	1.1
42	1.9	2.1	1.4	1.5	1.2	1.2	1.1	1.1	1.0	1.1

Table 2. Observed v. expected overlap. ***Obs*** **= measured overlap.** E_O **= expected overlap calculated from equation 1 assuming square bounding boxes (**$w = h$**)**

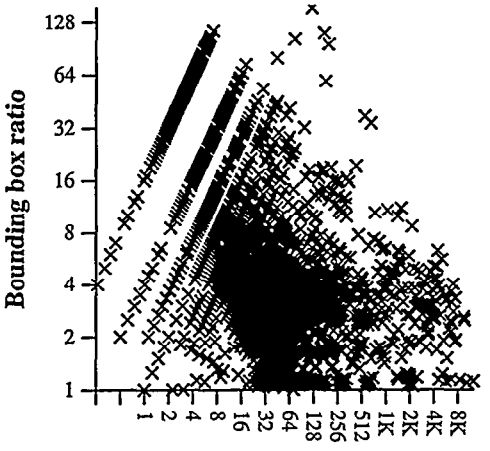


Figure 8. Scene 41.

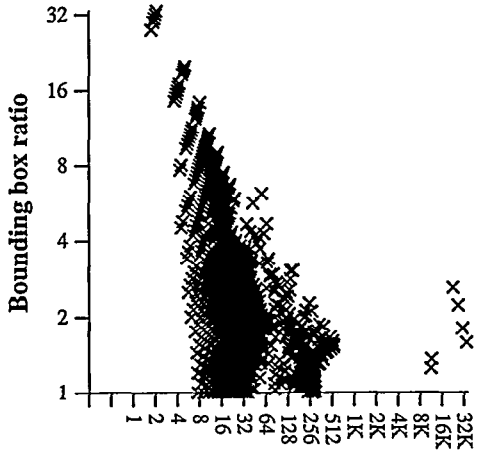


Figure 9. Scene 42.

6.3 Discussion

Contrary to expectation, our results suggest that in practice overlap is insensitive to bounding box aspect ratio when the bucket sort is by bounding box. We have found this result surprising in light of the observation (equation 2) that expected overlap is at a minimum when aspect ratio is 1.0. While we expect from equation 2 that aspect ratio should have less impact when bounding boxes are larger than a bucket, the scatter plots in Figures 2 through 9 make it clear that there are substantially many bounding boxes smaller than, say, a 32x32 bucket.[3] We conjecture that the reason for the discrepancy between observed and expected is that underlying distribution of $\sqrt{r}$ has a mean close to 2.0 for the scenes we have studied.[4] However, since our goal in this work has been the impact of bucket rendering on the PC architecture, we have not investigated this question further.

On the other hand, our results do provide experimental corroboration for the hardware architect who calculates overlap of screen-aligned bounding boxes using equation 1 and the square root of average bounding box area. That is, the original Molnar/Eyles equation (1) seems a reliable analytical tool for predicting overlap by letting $s = w = h = \sqrt{A}$.

6.4 Scaling with screen resolution

A last aspect of overlap requires discussion. Most treatments of overlap have ignored screen resolution. In PC graphics accelerator hardware, however, we expect bucket size to be fixed by the size of the on-chip memory dedicated to bucket rendering. Furthermore, scenes will not be redesigned for each resolution a given chip may support (and at which the user may choose to render).[5] As a result, bounding box area A scales with screen resolution. If A is measured at resolution R, then we take bounding box area A' at resolution R' to be $(R'/R)A$.

7 Architectural Implications of Bucket Size

In order to examine the effect of overlap on host memory, AGP, and triangle set-up on the graphics accelerator, we have done the following:

- From equation 1, we have calculated expected overlap for the eight scenes of Table 1. We have taken average bounding box area from that table, and let $s = \sqrt{A}$. In particular, we have ignored aspect ratio of bounding boxes.
- We have made the assumption that each primitive is written by the host to each bucket into which it lands, and that each primitive is read from each bucket by the accelerator.[6]

[3] This is numerically corroborated by Table 1, which shows the fraction of bounding boxes with area larger than 1K pixels.
[4] Please see the discussion of aspect ratio in section 6.1.
[5] The application may or may not even adjust field of view with change in resolution.
[6] We have not assumed "optimized" bucket storage based on pointers to primitives. If the accelerator DMAs from host memory, it is not clear that pointer-based buckets result in the best performance.

		Scene							
		14	*15*	*20*	*22*	*25*	*39*	*41*	*42*
640x480	*Overlap*	1.3	1.2	2.1	2.0	1.6	1.8	1.2	1.1
	Ktris (O)	33.1	24.0	109.5	41.4	128.3	18.4	188.2	205.1
	Ktris (raw)	25.3	19.4	52.8	20.3	80.5	10.4	156.3	182.6
	Mem (O)	5.6	4.0	18.4	7.0	21.6	3.1	31.6	34.5
	Mem (raw)	4.3	3.3	8.9	3.4	13.5	1.7	26.3	30.7
	AGP (O)	2.8	2.0	9.2	3.5	10.8	1.5	15.8	17.2
	AGP(raw)	2.1	1.6	4.4	1.7	6.8	0.9	13.1	15.3
1024x768	*Overlap*	1.5	1.4	2.9	2.8	2.0	2.3	1.3	1.2
	Ktris (O)	38.2	27.0	153.4	57.7	162.4	24.2	208.8	219.3
	Ktris (raw)	25.3	19.4	52.8	20.3	80.5	10.4	156.3	182.6
	Mem (O)	6.4	4.5	25.8	9.7	27.3	4.1	35.1	36.8
	Mem (raw)	4.3	3.3	8.9	3.4	13.5	1.7	26.3	30.7
	AGP (O)	3.2	2.3	12.9	4.8	13.6	2.0	17.5	18.4
	AGP(raw)	2.1	1.6	4.4	1.7	6.8	0.9	13.1	15.3
1280x1024	*Overlap*	1.7	1.5	3.6	3.6	2.4	2.8	1.4	1.3
	Ktris (O)	42.4	29.5	192.5	72.1	191.5	29.3	225.6	230.6
	Ktris (raw)	25.3	19.4	52.8	20.3	80.5	10.4	156.3	182.6
	Mem (O)	7.1	5.0	32.3	12.1	32.2	4.9	37.9	38.7
	Mem (raw)	4.3	3.3	8.9	3.4	13.5	1.7	26.3	30.7
	AGP (O)	3.6	2.5	16.2	6.1	16.1	2.5	18.9	19.4
	AGP(raw)	2.1	1.6	4.4	1.7	6.8	0.9	13.1	15.3

Table 3. S=128, Hz = 30.
***Overlap* calculated from equation 1. *Ktris* required with bucket rendering *(O)* and without *(raw)*. Memory bandwidth *(Mem)* required with bucket rendering *(O)* and without *(raw)*, in Mbytes/sec. *AGP* bandwidth required with bucket rendering *(O)* and without *(raw)*, in Mbytes/sec.**

- We have assumed isolated triangles (i.e. 3 vertices per triangle).
- We have assumed that each vertex comprises <x,y,z,u,v,w,rgba>, and that each of these fields requires 32 bits. This results in 28 bytes per vertex, 84 bytes per triangle.
- The calculations we have done assume 30 Hz frame rate. Higher frame rates scale directly from these bandwidth numbers.

The results appear in Tables 3 and 4. For each of the three resolutions in these tables, we have scaled bounding box area as discussed in section 6.4. The effect appears in the *Overlap* numbers at the target resolution. Following overlap are the triangle set-up throughput, and memory and AGP bandwidth that we calculate would be required for bucket rendering of the eight scenes in this paper. Triangle rates required of the accelerator set-up engine are shown in rows labeled *Ktris*; units are Ktriangles/sec. Memory bandwidth required is shown in rows labeled *Mem*; units are Mbytes/sec. AGP bandwidth required is shown in rows labeled *AGP*; units are Mbytes/sec. For each statistic we report two numbers, labeled by *O* and *raw*. *O* rows take into account overlap, and correspond to the throughput we calculate would be necessary for bucket rendering. *Raw* rows ignore overlap, and correspond to the throughput we calculate would be necessary for standard rendering (i.e. full-frame).

7.1 Discussion

We expect to see rendering engines in the near future that can perform set-up of 1M triangles/sec. The current sustainable delivered bandwidth from main memory on the PC is between 200 and 300 Mbytes/sec. We would like support for bucket rendering to consume only a fraction of this, since texturing from main memory requires significant bandwidth and since graphics is not the only memory bandwidth consumer. AGP is expected to deliver 256 Mbytes/sec of sustained bandwidth.

Consider Table 3 and the requirements when tiles are 128x128. All scenes show acceptable estimated requirements at all resolutions. Triangle set-up, memory bandwidth, and AGP bandwidth requirements stay below 25%, 15%, and 10% respectively. We conclude that with tiles 128x128 pixels in size, and bucket sorting by bounding box, there should be minimal impact of bucket rendering on main memory, AGP, and triangle set-up.

Consider now Table 4 and the requirements when tiles are 32x32 pixels. At 1024x768 one scene consumes a solid 40% of memory, another 30%. At 1280x1024 the same scenes consume >60% and > 45%. Furthermore, AGP bandwidth consumed at 1280x1024 can surpass 33%. These numbers are prohibitively high. We conclude from these results that when tiles are 32x32 pixels and bucket sorting is by bounding box, the performance impact of bucket rendering on main memory, AGP, and triangle set-up is minimal at 640x480, moderate at 1024x768 and quite high at 1280x1024.

		Scene							
		14	*15*	*20*	*22*	*25*	*39*	*41*	*42*
640x480	*Overlap*	2.5	2.1	7.6	7.4	4.2	5.4	1.9	1.5
	Ktris (O)	62.5	40.9	402.5	149.4	338.5	55.8	302.0	280.7
	Ktris (raw)	25.3	19.4	52.8	20.3	80.5	10.4	156.3	182.6
	Mem (O)	10.5	6.9	67.6	25.1	56.9	9.4	50.7	47.2
	Mem (raw)	4.3	3.3	8.9	3.4	13.5	1.7	26.3	30.7
	AGP (O)	5.2	3.4	33.8	12.5	28.4	4.7	25.4	23.6
	AGP(raw)	2.1	1.6	4.4	1.7	6.8	0.9	13.1	15.3
1024x768	*Overlap*	3.7	3.0	14.6	14.0	7.2	9.7	2.6	1.9
	Ktris (O)	92.7	57.7	769.5	284.0	578.6	100.4	412.2	349.6
	Ktris (raw)	25.3	19.4	52.8	20.3	80.5	10.4	156.3	182.6
	Mem (O)	15.6	9.7	129.3	47.7	97.2	16.9	69.3	58.7
	Mem (raw)	4.3	3.3	8.9	3.4	13.5	1.7	26.3	30.7
	AGP (O)	7.8	4.8	64.6	23.9	48.6	8.4	34.6	29.4
	AGP(raw)	2.1	1.6	4.4	1.7	6.8	0.9	13.1	15.3
1280x1024	*Overlap*	4.7	3.8	21.5	20.6	10.1	13.9	3.3	2.2
	Ktris (O)	120.2	72.7	1135.6	418.0	809.0	144.0	509.6	408.4
	Ktris (raw)	25.3	19.4	52.8	20.3	80.5	10.4	156.3	182.6
	Mem (O)	20.2	12.2	190.8	70.2	135.9	24.2	85.6	68.6
	Mem (raw)	4.3	3.3	8.9	3.4	13.5	1.7	26.3	30.7
	AGP (O)	10.1	6.1	95.4	35.1	68.0	12.1	42.8	34.3
	AGP(raw)	2.1	1.6	4.4	1.7	6.8	0.9	13.1	15.3

Table 4. S=32, Hz = 30.
***Overlap* calculated from equation 1. *Ktris* required with bucket rendering *(O)* and without *(raw)*. Memory bandwidth *(Mem)* required with bucket rendering *(O)* and without *(raw)*, in Mbytes/sec. *AGP* bandwidth required with bucket rendering *(O)* and without *(raw)*, in Mbytes/sec.**

These results lead to several conclusions:

- With sufficient on-chip memory to support 128x128 tiles, we expect the performance impact of bucket rendering to be minimal at resolutions to 1280x1024. This conclusion makes embedded DRAM an attractive technology for implementation of bucket rendering.
- With sufficient memory only to support 32x32 tiles, we expect acceptable performance at 640x480 with bounding box bucket sorting, but performance to degrade by 1280x1024. We expect on-chip SRAM to support only 32x32 tiles.
- For bucket rendering with 32x32 tiles, it is clear that for resolutions higher than 1024x768, alternatives to bucket sorting by bounding box will be desirable if not required. Possible alternatives have been mentioned in section 3. We cannot address the efficacy of these alternatives (in particular when host sort overhead is included). However, recall Table 1 and the fraction of bounding boxes larger than 1K pixels in area (i.e. larger than a 32x32 tile). Some of the scenes with high bucket rendering overhead (Table 4) certainly appear to be candidates for hybrid algorithms that perform exact sorting only on large primitives (e.g. Scenes 20 and 22). However, it also appears from this table that not all scenes might benefit from such an approach (e.g. Scene 25).

8 Summary

In this paper we have considered the impact on the PC architecture of accelerator support for bucket rendering. The key statistic upon which we have focused is primitive *overlap*. When a primitive overlaps multiple buckets, it must be placed in each. For each bucket that a primitive must be placed in, it must be written to and read from memory, must be sent across AGP, and must be set up by the accelerator for rasterization. We have explored the behavior of the most common bucket sorting algorithm – bucket sorting by bounding box, and have explored the architectural impact on the PC of accelerators that support one of two tile sizes: 32 x 32 pixels and 128 x 128 pixels. The former is approximately the size possible with dense SRAM, and typical PC price constraints. The latter is approximately the size possible with DRAM embedded in a logic process. We conclude that with 128 x 128 tiles, there is minimal impact on the PC architecture, with 32 x 32 tiles the impact may substantially degrade performance for resolutions higher than 1024x768. These results suggest that for architectures that employ SRAM for on-chip bucket rendering (and support tiles 32 on a side), further work is required to demonstrate bucket sorting algorithms that result in substantially less overlap (and overhead) than bounding box bucket sorting.

In the process of exploring the architectural implications of bucket rendering, we have extended results previously available on analytic expressions for overlap. In particular, our

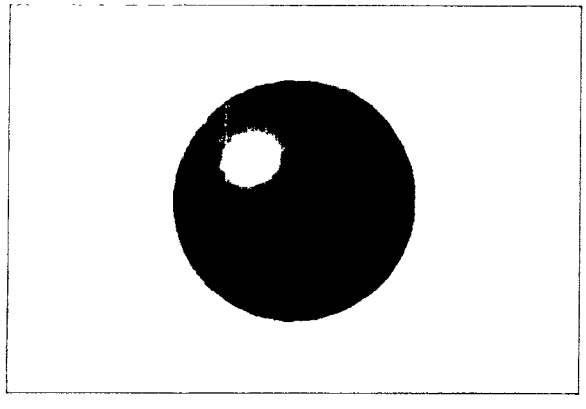
experiments suggest that bounding box ratio can be ignored when using the Molnar/Eyles equation for bounding box overlap, and in particular that the square root of the expected bounding box size can be employed directly in equation 1 that appears in section 5.

9 Acknowledgments

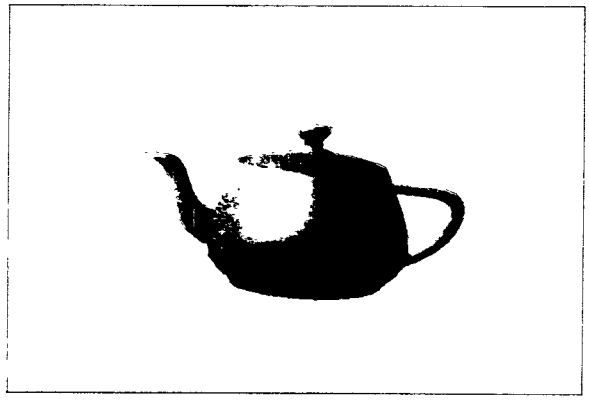
The authors would like to thank Dave Sprague, Jim Hurley, and Tim Misner for supporting this work, Ram Nalla and Feng Xie for help with rendering engine and ISM modifications. The first author would also like to thank Sam Uselton for helpful review and discussion of an earlier mathematical model of overlap with varying aspect ratio. The authors thank John Garney for very useful discussion about bucket sorting algorithms and hardware tile sizes. Many thanks to Evans & Sutherland for the Village database and Numerical Design Limited for the terrain.

10 References

1. K. Akeley, "RealityEngine Graphics," *Computer Graphics* (Proc. Siggraph), Vol. 17, No. 3, August 1993, pp. 109-116.
2. M. Cox, *Algorithms for Parallel Rendering*, Ph.D. thesis, Princeton University, May 1995.
3. D. Ellsworth, *Polygon Rendering for Interactive Visualizations on Multicomputers*, Ph.D. thesis, University of North Carolina at Chapel Hill, December 1996.
4. J. Foley, A. van Dam, S. Feiner, J. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed., Addison-Wesley, Reading MA, 1990.
5. H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Teggs, L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics* (Proc. Siggraph), Vol. 23, No. 3, July 1989, pp. 79-88.
6. Intel Corporation, *Accelerated Graphics Port Interface Specification*, revision 1.0, Intel Corporation, July 31, 1996.
7. S. Molnar, J. Eyles, and J. Poulton, "PixelFlow: High-Speed Rendering Using Image Composition," *Computer Graphics* (Proc. Siggraph), Vol. 26, No. 2, July 1992, pp. 231-240.
8. M. Kelley, S. Winner, K. Gould, "A Scaleable Hardware Render Accelerator using a Modified Scanline Algorithm," *Computer Graphics* (Proc. Siggraph), Vol. 26, No. 2, July 1992, pp. 241-248.
9. S. Molnar, *Image-Composition Architectures for Real-Time Image Generation*, Ph.D. thesis, University of North Carolina at Chapel Hill, October 1991.
10. M. Shantz, D. Krasnov, A. Kibkalo, A. Subbotin, F. Xie, and T. Park, "Building Online Virtual worlds," *Graphicon-96*, July 1-5 1996, GRAFO Computer Graphics Society, State Education Center, Saint Petersburg, Russia.
11. J. Torborg and J. T. Kajiya, "Talisman: Commodity Realtime 3D Graphics for the PC", *Computer Graphics* (Proc. Siggraph), August 1996, pp. 353-363.
12. S. Upstill, *The RenderMan Companion*, Addison-Wesley, Reading MA, 1989.

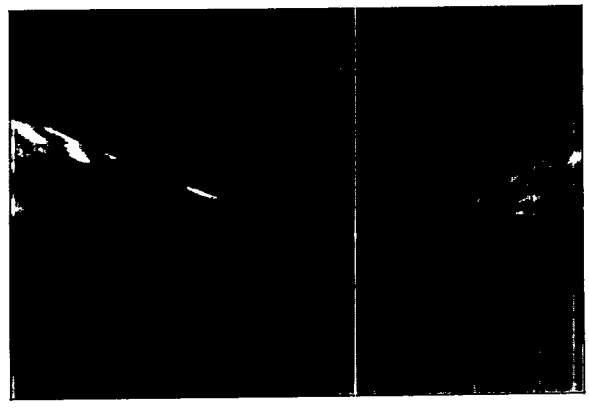


Screen 14 - Sphere



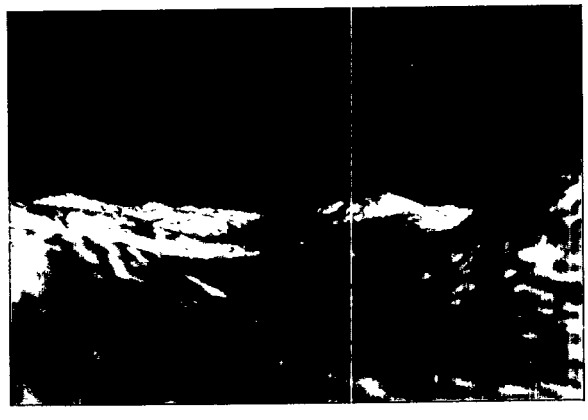
Screen 15 - Teapot

Screen 20 - Village
Courtesy: Evans & Sutherland

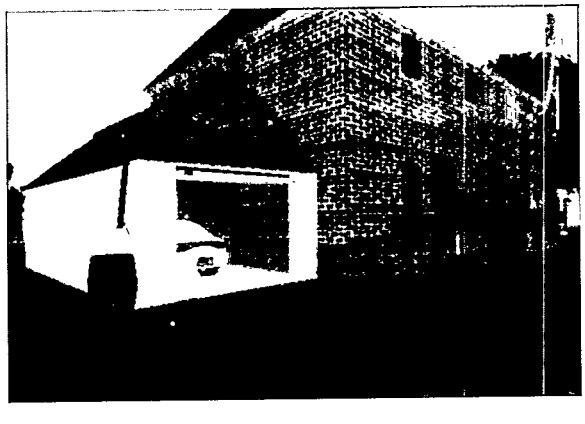


Screen 39 - Terrain
Courtesy: Numerical Design Limited

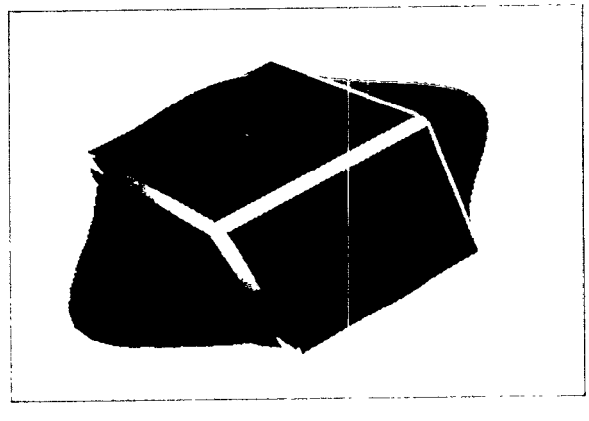
Screen 22 - Village
Courtesy: Evans & Sutherland



Screen 41 - Terrain
Courtesy: Numerical Design Limited



Screen 25 - Village
Courtesy: Evans & Sutherland



Screen 42 - Bezier Patches