Simple Models of the Impact of Overlap in Bucket Rendering

Milton Chen, Gordon Stoll, Homan Igehy, Kekoa Proudfoot and Pat Hanrahan

Computer Systems Laboratory

Stanford University

Abstract

Bucket rendering is a technique in which the framebuffer is subdivided into coherent regions that are rendered independently. The primary benefits of this technique are the decrease in the size of the working set of framebuffer memory required during rendering and the possibility of processing multiple regions in parallel. The drawbacks of this technique are the cost of computing the regions overlapped by each triangle and the redundant work required in processing triangles multiple times when they overlap multiple regions. Tile size is a critical parameter in bucket rendering systems: smaller tile sizes allow smaller memory footprints and better parallel load balancing but exacerbate the problem of redundant computation.

In this paper, we use mathematical models, instrumentation, and trace-driven simulation to evaluate the impact of overlap and conclude that the problem of overlap is limited in scope. If triangles are small, the overlap factor itself is also small. If triangles are large, overlap is high but pixel work dominates the rendering time. In pipelined rendering systems, the worst-case impact of overlap occurs when the area of an input triangle is equal to the area for which the pipeline is balanced—that is, the triangle-related computation time is equal to the pixel-related computation time. Thus, as the current trends of exponentially increasing triangle rate, slowly increasing screen resolution, and increasing per-pixel computation continue to push this balance point toward triangles with smaller area, bucket rendering systems will be able to utilize smaller tiles efficiently.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture.

1 INTRODUCTION

Bucket rendering is a technique in which the framebuffer is subdivided into coherent regions that are rendered independently. This technique has two primary benefits: a reduced framebuffer memory working set and the ability to process multiple regions in parallel. Standard rendering requires random access to framebuffer data for the entire screen, implying a memory system that is simultaneously large and fast. Bucket rendering requires random access to only a single tile, and smaller tiles are better as they lead to smaller (and thus faster) working memories. Additionally, bucket rendering allows for the possibility of rendering into a working set memory that is much deeper than the framebuffer itself: the tile working set memory can hold color, transparency, depth, stencil, etc., for multiple samples while the framebuffer retains only the display color. Bucket rendering also introduces image-space parallelism to the rendering pipeline and allows for multiple rasterizers that can render into independent tiles simultaneously. One advantage of such a subdivision is the fact that each triangle does not necessarily have to be processed by every rasterizer. In a tile-based parallel rasterizer, smaller tiles have the advantage of improving rasterization load balancing. The primary drawback of bucket rendering with small tiles is the large number of tiles overlapped by each triangle. As overlap increases, the work required to sort triangles into the appropriate tiles increases. More importantly, the fixed per-triangle work associated with rendering a triangle into a tile must be performed repeatedly.

The goal of this paper is to show that the impact of overlap on systems using small tiles is limited despite high overlap factors. We develop two simple models of the efficiency of bucket rendering, focusing on the impact of redundant work due to overlap. We first analytically model the per-pixel and per-triangle computations performed after bucket sorting, including triangle setup, scan conversion, texturing, depth buffering, and composting. Then, we present quantitative data on the overhead of tiling and compare it with the predictions of the analytic model. We conclude that the problem of overlap is limited in scope. If triangles are small, the overlap factor itself is also small. If triangles are large, overlap is high but per-pixel work dominates rendering time. In systems where triangle work and pixel work are pipelined, the worst-case impact of overlap occurs when the area of an input triangle is equal to the area for which the pipeline is balanced—that is, the triangle-related computation time is equal to the pixel-related computation time. Thus, as the current trends of exponentially increasing triangle rate, slowly increasing screen resolution, and increasing per-pixel computation continue to push this balance point toward triangles with smaller area, bucket rendering systems will be able to utilize smaller tiles efficiently.

2 RELATED WORK

Bucket rendering, also known as tiled rendering or chunking, is a feature of many previous systems including RenderMan [12], Talisman [11], PixelPlanes 5 [4], PixelFlow [10], and two systems developed at Apple [6][7]. Tiling in RenderMan serves to limit the large framebuffer memory working set required for oversampling, true transparency, CSG, motion blur, and other advanced features. In Talisman, the Apple systems, PixelPlanes 5, and PixelFlow, tiling is used to fit the working set into on-chip memory. In PixelPlanes 5 and PixelFlow, multiple tiles are also processed simultaneously by independent rendering engines.

Bucket rendering incurs redundant work that is characterized by the *overlap factor* (the number of tiles a triangle covers). Most systems approximate the tiles that a triangle overlaps by computing the intersection of the triangle's bounding-box with the grid of

{miltchen,gws,homan,kekoa,hanrahan}@graphics.stanford.edu

Terminology

Term	Definition
a	The area of a triangle, in pixels.
ρ	The ratio of the area of the bounding-box of a triangle to the area of the triangle.
S	The length of the side of a screen tile (assumed to be square), in pixels.
O	The *overlap factor* that represents the average number of tiles a triangle (or its bounding-box) overlaps.
k	The ratio of the processing time for a triangle to the processing time for a pixel.
R	The inefficiency due to overlap: the ratio of rendering time with tiling to rendering time without tiling.

Figure 1: Terminology used in the models.

tiles [1][3][4][10][11]. Molnar [8] presents an equation for the expected overlap of rectangular bounding-boxes on rectangular tiles that is experimentally verified in both Molnar [9] and Cox [2]. Sorting triangles into tiles exactly, rather than by bounding-box, is certainly possible and would result in lower overlap factors, but we are not aware of an analysis of the costs and benefits of such a technique or of any systems that use it.

Previously, the impact of bucket rendering has been analyzed by Cox [2] on a specific PC-based bucket rendering architecture. The paper concludes that for small tiles (32x32 pixels), the redundant storage and transfer of triangles due to high overlap factors can overwhelm current PC memory and I/O systems. In that work, the impact of triangle overlap is analyzed as it pertains to the architecture and bandwidths of a particular system. Our work focuses on the fundamental relationship between the overlap factor and the relative costs of per-triangle and per-pixel work within a general rendering framework.

3 ANALYTIC MODELS

In this paper, we model the pixel and triangle processing performed in relation to rasterization (i.e., after the object-space triangle has been transformed and sorted into the appropriate tiles).

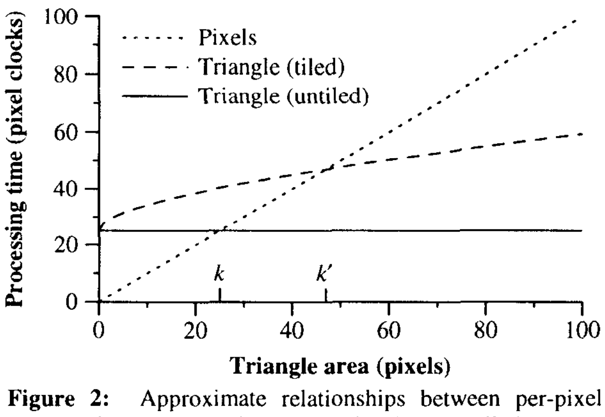


Figure 2: Approximate relationships between per-pixel computation, per-triangle computation in an untiled system, and per-triangle computation in a tiled system (the original per-triangle computation multiplied by the overlap factor). Assumes k set to 25 and S set to 32.

The key terms used in our analytic model are presented in Figure 1. The computation required for a pixel is assumed to be constant, though our model could be extended to analyze the rendering of scenes with a mix of rendering modes on systems with varying fill rates. The computation required for processing a single triangle in a non-tiled system is also assumed to be constant. The per-pixel cost is normalized to one, and the per-triangle cost, denoted by k, is normalized to a multiple of the per-pixel cost. In a bucket rendering system, when a triangle overlaps multiple tiles, its per-triangle costs are multiplied by the overlap factor, O. Figure 2 shows the simple relationships between triangle area, overlap factor, and the costs associated with per-pixel work and per-triangle work. We use this graph as a baseline for creating two models of a rasterization system, a *software model* and a *hardware model*.

The software model abstracts a system with a single unit for both triangle and pixel processing. The hardware model is motivated by the fact that many systems perform triangle and pixel processing on independent units in a pipeline. Figure 3 shows the relationship of triangle area to processing time on several such systems: the Silicon Graphics O2, RealityEngine, and InfiniteReality. When triangles are small, the processing times are fixed. When triangles are large, the processing times are directly proportional to triangle area. As we will show, these characteristics significantly change the effect of overlap on pipelined systems when compared to non-pipelined systems.

3.1 Modeling Overlap

Overlap of multiple tiles by a single triangle is characterized by the overlap factor, O, and is central to evaluating the efficiency of bucket rendering. Due to the high cost of calculating the exact tiles overlapped by a triangle, we instead calculate overlap using axis-aligned bounding-boxes. The exact overlap factor for a specific triangle can be calculated easily, but for general analytic calculations, the Molnar-Eyles equation [8] can be used to calculate approximate bounding-box overlap. The equation approximates the expected overlap factor of a triangle by utilizing S (the dimension of a tile), a (the triangle area), and ρ (the ratio of the area of a triangle's bounding-box to the area of the triangle):

$$O = \left(\frac{S + \sqrt{\rho a}}{S}\right)^2 \qquad (1)$$

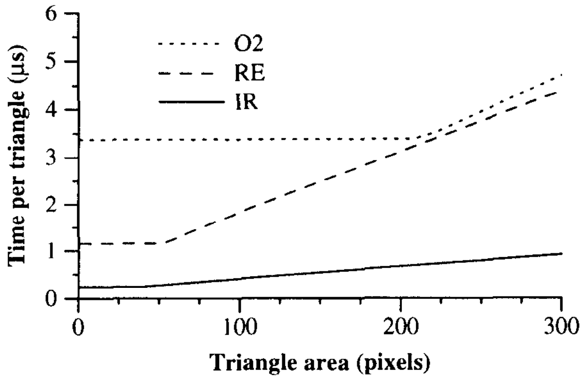


Figure 3: The graph above shows the average rendering time per triangle as triangle area changes on an SGI O2, RealityEngine, and InfiniteReality. Triangles are isolated, axis-aligned, right isosceles, flat-shaded, and untextured with depth buffering and blending disabled.

The primary source of error in this approximation is the assumption of unit aspect ratio (i.e., a square bounding-box). Furthermore, if we interpret Equation 1 to account for a set of triangles in a scene rather than a single triangle, we must assume that all of the bounding-box areas are the same. Unit aspect ratio is a best case for overlap [2], while uniform triangle area is a worst case (see the Appendix for a derivation). The two effects tend to offset each other in the simple form of the equation above, and adjusting the equation to account for one and not the other generally results in worse agreement with measured overlap. Analytic calculations in this paper use the simple form above, and use the approximation $\rho = 3$ (as justified in [8]).

3.2 Software Model

In our software model of rasterization, a single processing unit is responsible for both the per-triangle work and the per-pixel work associated with the rasterization of a triangle. Thus, the total processing time required for a triangle is equal to the sum of the per-triangle and per-pixel times. With our normalization of the per-pixel cost to one, the total processing time for a triangle is simply $k+a$. In a tiled system, the per-triangle cost is incurred O times, making the total processing time $kO+a$. Thus, the expected ratio of the processing time of a triangle with tiling to that without tiling is:

$$R_{sw} = \frac{SUM(kO,a)}{SUM(k,a)} = \frac{kO+a}{k+a} \tag{2}$$

Figure 4 shows R_{sw} for $k = 25$ and $S = 32$, with overlap calculated analytically using Equation 1. We define a_{worst} to be the value of a which maximizes R_{sw}; this is the triangle area that results in the worst efficiency. The formula for deriving this value can be found in the Appendix. A formula for the limiting value of R_{sw} as a approaches infinity is also given in the Appendix. Equation 2 describes the inefficiency ratio of tiling for a single triangle. However, because the terms involved are linear, we can use the same equation for calculating the inefficiency ratio for a distribution of triangles by interpreting a as the average triangle area and O as the average overlap.

3.3 Hardware Model

The hardware model of rasterization abstracts a system in which triangle and pixel processing are performed by independent processing units in a pipeline. Furthermore, we assume perfect pipelining between these two units, meaning that an infinite amount of buffering exists between the two. This buffering decouples the pixel work from the specific triangle that generated it; thus, the total processing time for a triangle is the maximum of the total time spent in per-pixel processing and the total time spent in per-triangle processing. We experimentally evaluate the effects of the infinite buffering assumption in Section 4.3.

In Figure 2, the curve for pixel processing intersects the curves for triangle processing with and without tiling, dividing the range of triangle areas into three regions. For triangles with area less than k, triangle processing dominates both the tiled and untiled cases. For triangles with area between k and k', triangle processing dominates the tiled case while pixel processing dominates the untiled case. For triangles with area greater than k', pixel processing dominates both cases. The value of k' is derived in the Appendix, and the expected ratio of the processing time of a single triangle with tiling to that without tiling is given by:

$$R_{hw} = \frac{MAX(kO,a)}{MAX(k,a)} = \begin{cases} O & 0 < a \le k \\ \frac{kO}{a} & k \le a \le k' \\ 1 & k' \le a \end{cases} \tag{3}$$

Figure 5 shows R_{hw} with overlap calculated analytically using Equation 1. The triangle area with the lowest efficiency (a_{worst}) occurs at k pixels, independent of the tile size. This property of the hardware model is derived in the Appendix. Equation 3 was derived for a single triangle, but we can generalize it to a distribution of triangles because the assumption of infinite buffering decouples the pixel work from any specific triangle. In this case, a is simply the average triangle area of the triangle distribution, and O the average overlap.

The key point of Equation 3 and Figure 5 is that the inefficiency of tiled rendering due to triangle overlap is limited in scope. For large triangles, pixel costs dominate. For small triangles, the overlap factor is small, and thus the inefficiency is small. Interestingly, the worst case behavior for the hardware model occurs at precisely the triangle area for which the system is balanced between triangle processing and pixel processing (k pixels). Furthermore, this worst case is independent of the tile size.

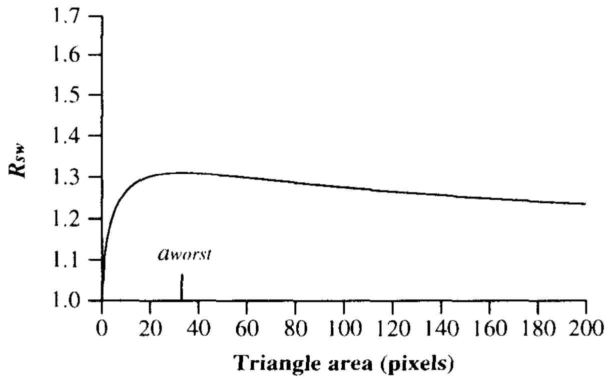


Figure 4: The graph above shows the expected ratio of rasterization time with tiling to that without tiling for a software system (R_{sw}) as triangle area (a) changes. The values for the graph are computed from Equation 2 with k set to 25, ρ set to 3, and S set to 32.

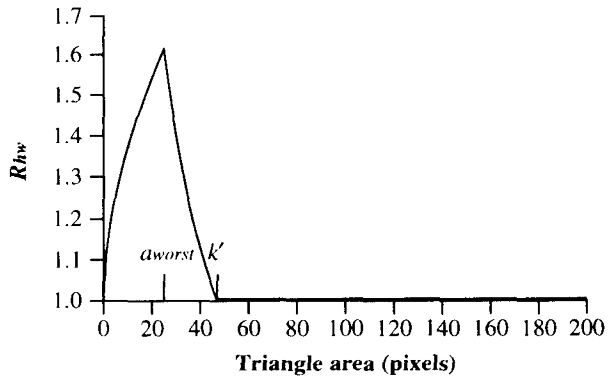


Figure 5: The graph above shows the expected ratio of rasterization time with tiling to that without tiling for a hardware system (R_{hw}) as triangle area (a) changes. The values for the graph are computed from Equation 3 with k set to 25, ρ set to 3, and S set to 32.

	Input Triangles	Visible Triangles	Avg. Area	Depth Cmplx.	Shading
Quake	1138	536	2784.4	1.90	Texture
Flight	3100	643	1544.3	1.26	Texture
Studio	15036	1676	807.8	1.72	Gouraud
Cylhead	11802	4836	38.3	0.24	Gouraud
Head	59592	29045	8.4	0.31	Gouraud

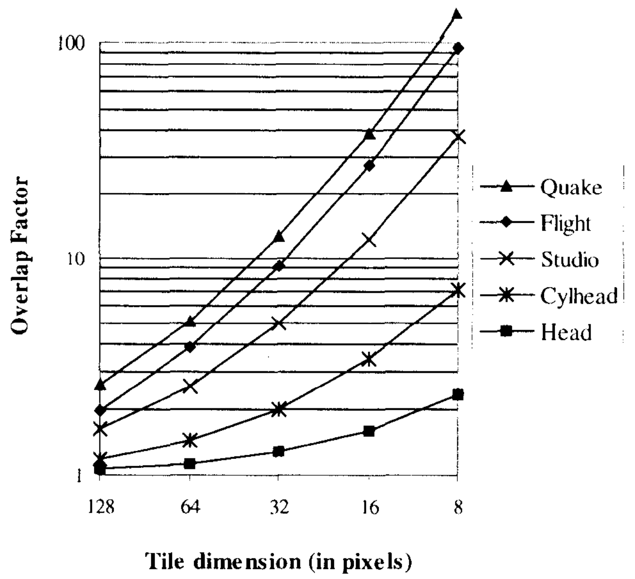


Figure 6: Statistics for the five test scenes used to evaluate our models are shown in the table above. The bounding-box overlap factors of these scenes (which were measured on a triangle-by-triangle basis) are shown in the graph.

4 EXPERIMENTAL EVALUATION

In Section 3, we provided analytic calculations that characterize the overhead associated with overlap in a bucket rendering system. We now present experimental data on this overhead in order to confirm the predictions of the analytic models and expose the shortcomings of the various assumptions that were made in formulating the equations.

4.1 Datasets

In order to gather relevant real-world experimental data, we captured scenes by tracing OpenGL programs with an OpenGL Stream Codec (GLS) [13] tracing tool. One frame from each of five test scenes is used in this study, two with texture and three without texture. These scenes, summarized in Figure 6, were rendered at 1024x768 screen resolution with varying tile sizes. Pictures of these frames can be found at the end of this paper. These scenes were selected due to the wide variation in the average triangle size. This leads to a wide range of overlap factors: Figure 6 also shows the exact bounding-box overlap factor for these scenes for a variety of tile sizes. These values are derived by summing the number of tiles overlapped by the bounding-box of each individual triangle and dividing by the total number of triangles. At one extreme, Head has an overlap of only 2.4 even when there are 12,288 tiles (8x8 tiling). At the other extreme, Quake has an overlap of 132 at that same tiling.

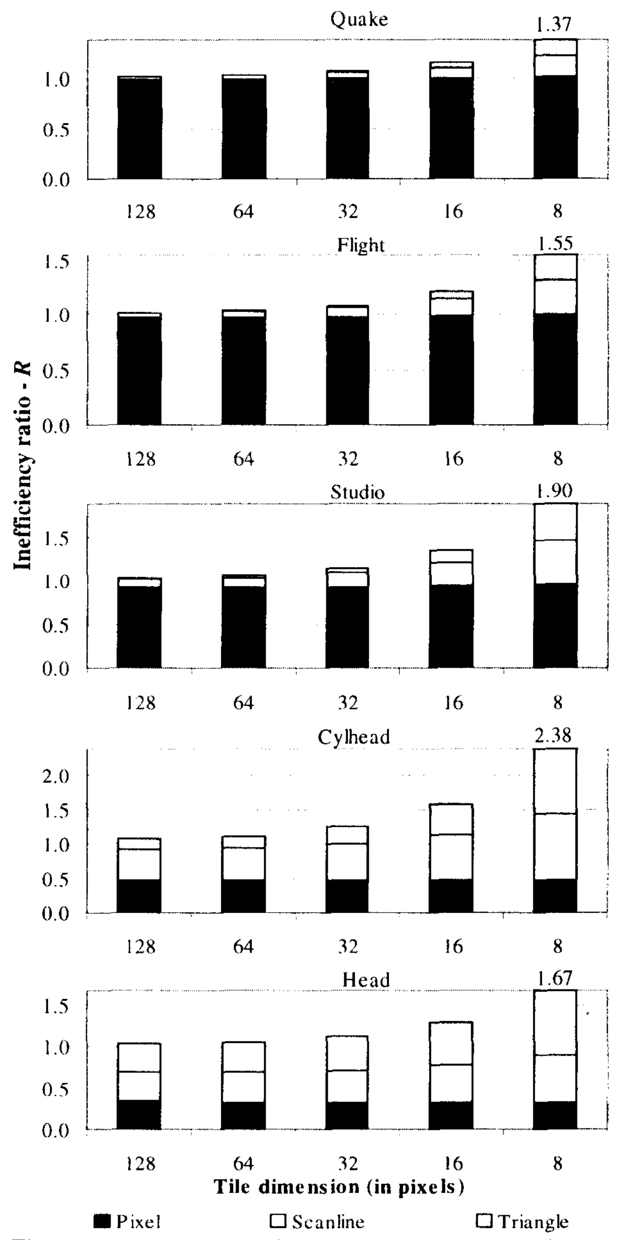


Figure 7: Measured *R* of software rasterization in Argus. The three categories are per-pixel work, per-scanline work and per-triangle work. Per-triangle work includes triangle setup.

4.2 Software Model

The computational overhead of tiling for a uniprocessor software renderer was measured by rendering the test scenes through our research rendering system *Argus*. In order to gather accurate, fine-grained timing information, Argus was run on top of the full machine simulator *SimOS* [5]. The instrumentation capabilities of SimOS allowed measurements to be taken at a per-pixel granularity without altering timing behavior.

Figure 7 shows the ratio of rasterization time with tiling to that without tiling for each of the test scenes, as measured in Argus. The rasterization time is separated into per-pixel, per-scanline, and per-triangle categories. Computation time which could not be

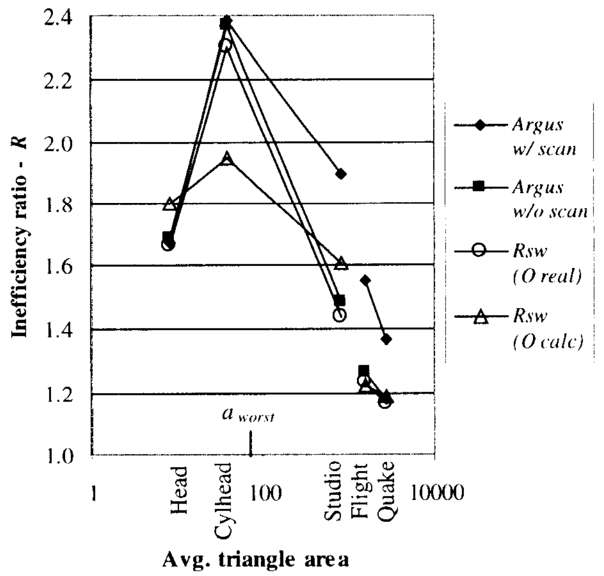


Figure 8: Comparison of the inefficiency as measured in the Argus renderer and as calculated from the software model using both measured and calculated overlap. a_{worst} equals 70 for non-textured scenes. The curves for the two k's (textured and non-textured) are separated. All data is for an 8x8 tiling.

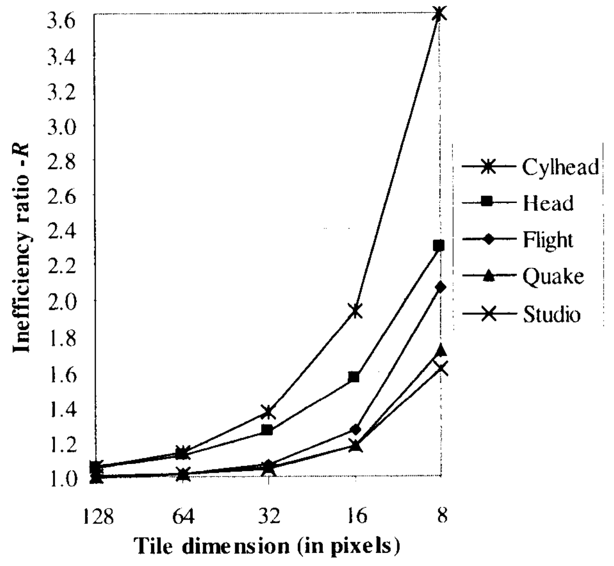


Figure 9: Observed R for a simulated rasterizer with a single triangle of buffering between pipelined triangle and pixel processors. Simulated for $k = 25$.

easily assigned to one of these categories was measured, but was less than four percent in all cases and is not included in the figure.

Processing times for Quake, Flight, and Studio are dominated by pixel processing, limiting the effect of overlap. For Cylhead and Head, scanline and triangle processing dominate. Note that in all scenes, per-pixel processing is independent of the tile size. All of the test scenes have significantly better efficiency than their overlap alone would indicate. Rendering efficiency is close to 100% for all scenes with 128x128 tiles. The best efficiency at 8x8 tiles is 1.37 for Quake, which has the highest overlap at 132. The worst efficiency is 2.38 for Cylhead, with an overlap of 7.2. As expected from the model, the worst efficiency occurs for the scene with triangles of moderate size.

In order to quantify the effect of approximations used in the model, we need k, O and a for use in the R_{sw} calculation in Equation 2. Two values of k for Argus were measured from test data, one for non-textured rendering ($k = 9.3$) and one for textured rendering ($k = 3.6$). The overlap is derived in two ways: O_{real} uses data from measured bounding-box overlap, and O_{calc} uses the Molnar-Eyles equation. The average triangle area of each scene is used for a. Figure 8 shows R_{sw} calculated with the two overlap factors along with the inefficiency ratios measured from Argus. $Argus_{w/scan}$ is the same data shown in Figure 7. $Argus_{w/o\ scan}$ is the same data without the scanline costs.

Figure 8 shows that each of the derivations of R_{sw} generate qualitatively similar curves. All curves peak at Cylhead and drop off sharply on either side of Cylhead. However, the software model has significant limitations. Comparing $R_{sw}(O_{real})$ and $R_{sw}(O_{calc})$ to $Argus_{w/o\ scan}$, notice the large gap between $R_{sw}(O_{calc})$ and $Argus_{w/o\ scan}$, for Cylhead. This gap is due to the error from the Molnar-Eyles overlap factor approximation (Equation 1). Comparing $Argus_{w/\ scan}$ and $Argus_{w/o\ scan}$ to $R_{sw}(O_{real})$, notice the deviation of $Argus_{w/\ scan}$ from other curves for scenes Studio, Flight, and Quake. This gap shows that the lack of a scanline processing factor is also a significant source of error in the analytical model of Section 3.2.

4.3 Hardware Model

In the analytic equations of Section 3.3, we assumed that a hardware rasterizer consisted of a triangle unit and a pixel unit separated by an infinite amount of buffering. In order to evaluate the overhead of tiling for a hardware rasterizer without the assumption of perfect pipelining, we simulated a system with only a single triangle of buffering between the triangle and pixel processing stages. For each triangle, for each tile overlapped by the triangle, the number of pixels in the tile which are covered by the triangle is counted. The rendering of these pixels is pipelined with the triangle processing for the following triangle, so the maximum of k and the number of covered pixels is counted toward the total rendering time.

Figure 9 shows the ratio of the simulated rasterization time with tiling to that without tiling for each of the test scenes for $k = 25$. Rendering efficiency is close to 100% for all scenes at 128x128 pixel tiles. The best efficiency at 8x8 pixel tiles is 1.62 for Studio, with an overlap of 37. As expected from the model, the scene with the worst efficiency is again Cylhead at 3.63.

In order to quantify the effect of approximations used in the model of Section 3.3, we need k, O and a for use in the R_{hw} calculations of Equation 3. The value of k is set to 25, as in the simulation. The overlap O is derived in two ways: from measured data (O_{real}) and from the Molnar-Eyles equation (O_{calc}). The average area of each scene is used for a. Figure 10 compares the inefficiency ratios measured using the simulator (Sim_{finite}) to those calculated using Equation 3 (R_{hw}). Both $R_{hw}(O_{real})$ and $R_{hw}(O_{calc})$ assume the buffer between the triangle and pixel stages can hold an unlimited number of triangles. Despite this simplifying assumption, our simple model predicts correctly that the inefficiency ratios will peak for scenes with triangle area close to k, as in Cylhead.

It is interesting to understand why Sim_{finite} is smaller than $R_{hw}(O_{real})$ for Cylhead but larger for Studio, Flight, and Quake in Figure 10. To understand this phenomenon, we must first look at

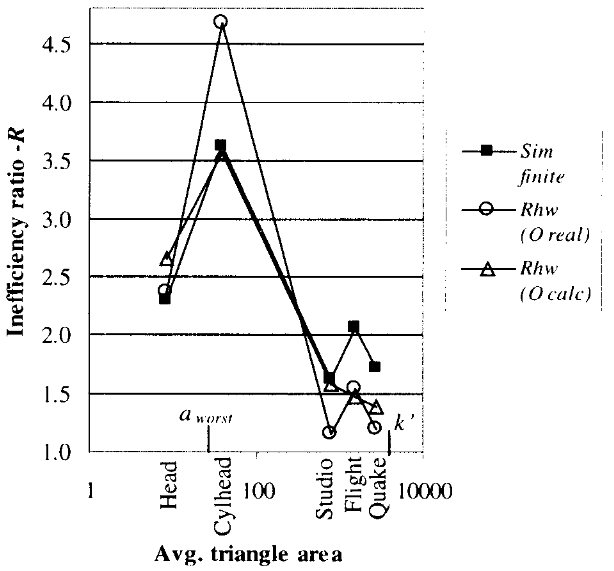


Figure 10: Comparison of the inefficiency of the simulated rasterizer and the inefficiency predicted by the hardware model using both measured and calculated overlap. k is set to 25 for both the simulation and the calculations. The value of a_{worst} is 25 and the value of k' is 3670. All data is for an 8x8 tiling.

how triangles with varying size are handled in each timing. The only difference between Sim_{finite} and $R_{hw}(O_{real})$ is the size of the buffer space between the triangle and pixel stages. In general, the pipeline will stall if the per-triangle stage cannot find a slot to store the intermediate data for a new triangle. More buffer space means that the pixel stage has a larger window of triangles with which to catch up to the triangle stage, thus avoiding some stalls. The exact relationship between this buffer size and rendering time depends on the distribution of triangles. In general, buffering is most effective when triangle area is close to k. For scenes with average triangle areas far greater than k, the pixel stage is likely to determine total processing time regardless of buffer size. For scenes with average triangle areas far less than k, the triangle stage is the limiting step, and buffering again has little effect.

The observations above are experimentally confirmed. Without tiling, the simulated processing time for Quake with finite buffering is 1.496 million cycles, while with infinite buffering it is predicted to be slightly lower at 1.492 million cycles. The change is small because the average triangle area (2784 pixels) is much greater than k (25 pixels). If we take tiling into account, when Quake is rendered with 8x8 tiles, the simulated processing time with a single triangle of buffering is 2.57 million cycles, while with infinite buffering, it is predicted to be 1.77 million cycles. Using an 8x8 tiling lowers the average effective triangle area seen by the rasterizer to 21 pixels, and infinite buffering significantly lowers the rendering time. This results in $R_{hw}(O_{real})$ being significantly lower than Sim_{finite} for Quake (Flight and Studio behave similarly). The same reasoning explains the reverse behavior observed for Cylhead in Figure 10. Cylhead benefits significantly from buffering without tiling, but 8x8 tiling moves the effective triangle area below the region where buffering is effective. This results in $R_{hw}(O_{real})$ being greater than Sim_{finite} for Cylhead.

The agreement of Sim_{finite} and $R_{hw}(O_{calc})$ for Cylhead is coincidental – the buffering effect described above closely matches the error in the Molnar-Eyles overlap for Cylhead. The small peak at Flight which is missing for $R_{hw}(O_{calc})$ is also due to overlap calculation error.

5 SUMMARY AND DISCUSSION

In this paper, we have evaluated simple models of the impact of overlap on the efficiency of rasterization in bucket rendering systems. The impact is shown to be highest for scenes in which triangle and pixel work are approximately balanced rather than for scenes with very large triangles (and correspondingly high overlap factors). In particular, the impact of overlap in pipelined systems is limited to a window of triangle sizes around the design point of the system. This in turn limits the magnitude of the observed overheads at levels far below the raw overlap factors.

The analytic models used in this paper are quite simple, and have a number of limitations. First, a system might have a significant amount of work associated with each scanline. Second, we assume that triangle meshing is not maintained through the bucket sorting stage. Rasterization accelerators that take advantage of meshing (e.g., for efficient communication of triangles) will suffer in a bucket rendering system if meshing is not maintained. Even though meshing can be maintained, vertex redundancy will occur at tile boundaries. Third, triangle and pixel operations are not necessarily independent. For instance, changing the processing order of triangles can affect texture locality and thus the cost of pixel processing. Fourth, the possible effects of tiling on graphics state change overhead is not modeled by our framework. Finally, the cost of sorting triangles into tiles is not currently modeled.

Our experiments focused on a rasterizer with $k = 25$, a design point we consider representative of current and near-future architectures. The behavior of systems at the extremes of this design space is instructive. A system with $k = 0$ performs all processing on a per-pixel basis. Such a system is insensitive to triangle overlap. A system with $k = S^2$ rasterizes a triangle into a tile in time independent of the number of pixels covered, as in the Pixel-Planes 5 system [4]. The efficiency of this system suffers in direct proportion to overlap.

It is interesting to consider the use of current pipelined rasterization accelerators, or similar devices, in bucket rendering systems. These rasterizers are being designed for smaller and smaller triangles. Given that the impact of overlap is greatest at this design point, the worst-case overlap factors will decrease. Designing for smaller triangles would seem to indicate that more chip resources should be dedicated to triangle processing. However, the cost of pixel processing is increasing rapidly due to features such as tri-linear mip-mapping, multi-texturing, and anti-aliasing. This decreases the relative impact of engineering the rasterizer for these smaller triangles, or even over-engineering it to accommodate overlap.

It is also interesting to consider the use of rasterization accelerators in *parallel* tiled rendering systems. Such systems can be highly scalable in both triangle rate and pixel rate, but rendering efficiencies degrade for two reasons. Inefficiencies due to the overlap factor of tiling, as described in this paper, limit the ideal speedup of a parallel system over a non-tiled system; this limit can be raised by utilizing larger tiles. Parallel tiled systems also suffer from spatial and temporal load imbalance. To combat this load imbalance, smaller tiles (with higher inefficiencies) may be used, or a dynamic load balancing algorithm may be employed. We are currently exploring the tradeoffs between dynamic tile scheduling and static assignment of small tiles, with the goal of creating scalable architectures based on tiled rasterization.

Acknowledgements

We would like to thank Yung-Hsiang Lu, Robert Bosch, and the SimOS group for supporting us through our SimOS effort. We thank Matthew Eldridge and John Owens for their reviews of this paper and Jingli Wang for deriving the equations of a_{worst} and k'. We thank Phil Lacroute for providing *glstrace*. Financial support was provided by Silicon Graphics, Intel, DARPA contract DABT63-95-C-0085-P00006, and the Department of Defense Graduate Fellowship with contribution from the US Air Force.

References

[1] M. Cox. *Algorithms for Parallel Rendering*, Ph.D. thesis, Princeton University, May 1995.

[2] M. Cox, N. Bhandari. Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC, *1997 Siggraph/Eurographics Workshop on Graphics Hardware*, pp. 25-34.

[3] D. Ellsworth. *Polygon Rendering for Interactive Visualizations on Multicomputers*, Ph.D. thesis, University of North Carolina at Chapel Hill, December 1996.

[4] 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.

[5] S. Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*, Ph.D. thesis, Stanford University, February 1998.

[6] M. Kelley, K. Gould, B. Pease, S. Winner, A. Yen. Hardware Accelerated Rendering of CSG and Transparency, *Computer Graphics (Proc. Siggraph)*, July 1994, pp. 177-184.

[7] 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.

[8] S. Molnar. *Image-Composition Architectures for Real-Time Image Generation*, Ph.D. thesis, University of North Carolina at Chapel Hill, October 1991.

[9] S. Molnar, M.Cox, D.Ellsworth, H. Fuchs. A Sorting Classification of Parallel Rendering, *IEEE Computer Graphics and Applications*, Vol. 14, No. 4, July 1994, pp. 23-31.

[10] 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. 241-248.

[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.

[13] http://trant.sgi.com/opengl/docs/Specs/glsspec.txt

APPENDIX

Worst-Case Overlap Factor

The expected number of tiles overlapped by two triangles of total bounding-box area $2B$ is:

$$O = \left(\frac{S+\sqrt{2B-b}}{S}\right)^2 + \left(\frac{S+\sqrt{b}}{S}\right)^2$$

The derivative of O is

$$O' = \frac{1}{S}\left(\frac{1}{\sqrt{b}} - \frac{1}{\sqrt{2B-b}}\right)$$

which is positive for $b < B$, zero at $b = B$, and negative for $b > B$. Therefore O is maximized at $b = B$, i.e., any two triangles of a given total area have maximum expected overlap if their areas are equal. This derivation can be generalized to conclude that any set of triangles has a maximum expected overlap if all of the triangle areas are equal.

Value of a_{worst}

In the software model, bucket rendering efficiency is described by Equation 2:

$$R_{sw} = \frac{kO + a}{k + a}$$

The overlap factor can be estimated with Molnar-Eyles equation:

$$O = \left(\frac{S + \sqrt{\rho a}}{S}\right)^2$$

The maximum value of R_{sw} can be derived by substituting the Molnar-Eyles overlap into Equation 2 and taking the derivative of the resulting expression. The triangle area where R_{sw} is maximum is:

$$a_{worst} = \left(\frac{k\rho + \sqrt{k^2\rho^2 + 4kS^2\rho}}{2S\sqrt{\rho}}\right)^2$$

In the hardware model, bucket rendering efficiency is described by:

$$\begin{aligned} R_{hw} &= O & 0 < a \le k && (3.1) \\ &= \frac{kO}{a} & k \le a \le k' && (3.2) \\ &= 1 & k' \le a && \end{aligned}$$

Equation 3.1 is a monotonically increasing function while Equation 3.2 is a monotonically decreasing function. Therefore, the triangle area where R_{hw} is maximum is:

$$a_{worst} = k$$

Value of k'

The triangle area at which the per-pixel processing curve and the tiled per-triangle processing curve intersects is indicated by k'. To find k', find the triangle area such that:

$$kO = a$$

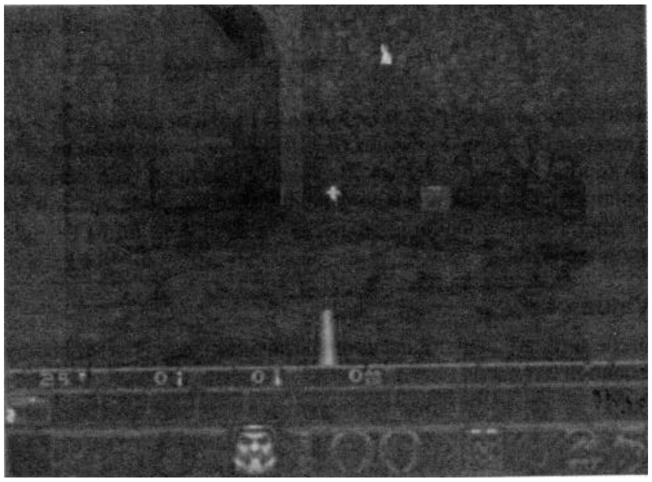
Using the Molnar-Eyles equation to estimate the overlap factor in the above equation, the triangle area of the intersection point is:

$$k' = \left(\frac{-kS\sqrt{\rho} - S^2\sqrt{k}}{k\rho - S^2}\right)^2$$

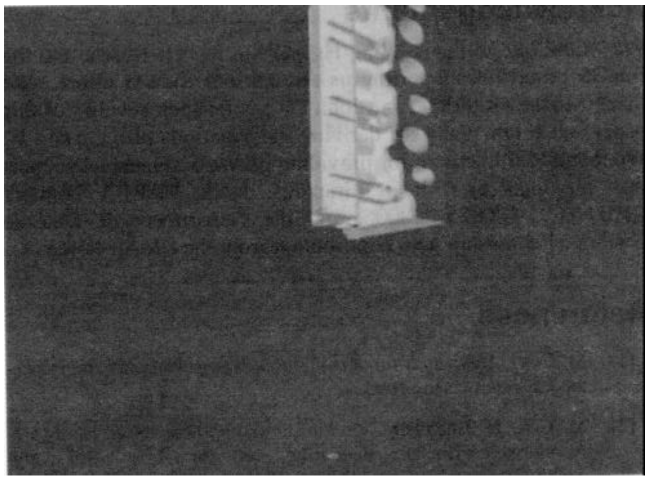
Value of R_{sw} as triangle area approaches infinity

Using the Molnar-Eyles overlap in Equation 2, and taking the limit as a approaches infinity, R_{sw} approaches:

$$R_{sw} = 1 + \frac{k\rho}{S^2}$$



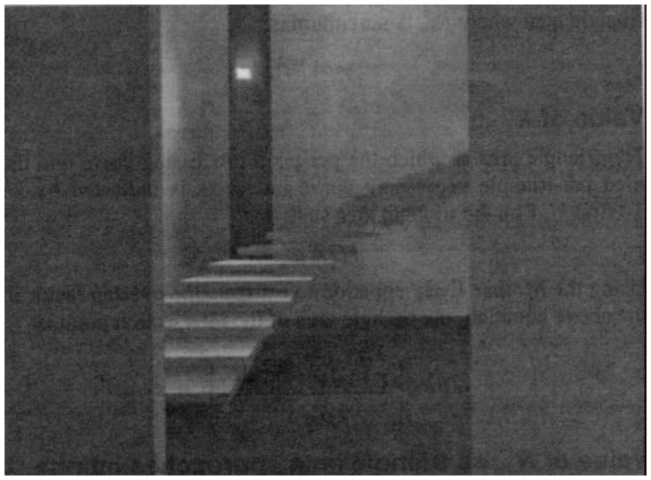
Quake – frame 22
Courtesy of id Software



Cylhead – frame 12
Courtesy of Picture-Level Benchmarks

Flight – frame 122
Courtesy of Silicon Graphics Reality Demo Suite

Head – frame 12
Courtesy of Picture-Level Benchmarks



Studio – frame 12
Courtesy of Picture-Level Benchmarks