Dynamic Algorithms for Sorting Primitives Among Screen-Space Tiles in a Parallel Rendering System

Rudrajit Samanta and Thomas Funkhouser
Princeton University

Abstract

Based on the recent improvements in the price/performance of PC-based graphics accelerators, it is compelling to consider approaches that combine multiple PCs together into a high performance, high resolution, low cost parallel rendering system. We are investigating sort-first approaches to build such a system in which one or more client PCs distribute 3D graphics primitives to multiple server PCs that render images for separate 2D tiles for projection on a display surface. A fundamental challenge in implementing such a system is developing a method to classify 3D graphics primitives according to their overlaps with the 2D screen-space tiles so that every server PC is not required to process every 3D primitive. In this paper, we describe several overlap classification algorithms based on different bounding volume approximations, different transformation sequences, different primitive aggregation methods, and different primitive sizes. We built a prototype sort-first system with one client PC and eight server PCs connected by a network to test these algorithms with the goal of characterizing their trade-offs and finding the one that maximizes the polygon throughput of the entire system. Interestingly, we find that the effectiveness of each overlap classification algorithm varies widely with different types of 3D scenes, and thus it is impossible to choose a single overlap classification algorithm that is best in all circumstances. Motivated by this observation, we incorporate dynamic adjustment of parameters of the overlap classification algorithm to balance the client and server processing times in real-time as 3D primitives are processed. With this new feature, we are able to achieve higher polygon rendering rates with our system.

1 Introduction

In the quest to develop a high-performance and low-cost polygon rendering system, it is attractive to consider the use of multiple PCs with fast, commodity graphics accelerator cards. PC graphics systems have improved at an astounding rate over the last few years, and their price-to-performance ratios far exceed those of traditional high-end rendering systems. This trend makes it attractive to consider approaches that combine the aggregate performance of many PCs connected together by a network into a high performance, high resolution, low cost parallel rendering system.

We believe it is possible to build such a system using a sort-first approach [28] in which one or more *client* PCs distribute 3D graphics primitives to multiple *server* PCs that render images for separate 2D tiles of a display. We are motivated to use sort-first to leverage the tight coupling between geometry and rasterization processors inside typical PC graphics accelerator cards, and to avoid interprocess communication that would be required to composite rendered images in a sort-last approach.

A fundamental challenge in a sort-first system is to develop algorithms that classify 3D graphics primitives according to their overlaps with 2D screen-space tiles. In this paper, we describe several overlap classification algorithms representing different trade-offs between speed and accuracy. Our investigation includes different bounding volume approximations, different transformation sequences, different primitive aggregation methods, and different primitive sizes. We executed experiments using a prototype sort-first system to characterize the trade-offs of different overlap classification algorithms and to find the algorithm that maximizes the polygon rendering throughput. Interestingly, we find that the effectiveness of each overlap classification algorithm varies widely with different types of 3D scenes, and thus it is impossible to choose a single algorithm that is best in all circumstances. Motivated by this observation, we incorporate dynamic adjustment of the overlap classification algorithm into our system to balance the client and server processing times in real-time as 3D primitives are processed. With this new feature, we are able to achieve higher polygon rendering rates.

The paper is organized as follows. The next section reviews related work in parallel polygon rendering, paying special attention to methods based on screen-space decompositions and systems utilizing a network of PCs. In Sections 3 and 4, we discuss the challenges of overlap classification and propose several algorithms. Section 5 contains a description of our prototype system, while Section 6 presents results of experiments with this system aimed at characterizing the effectiveness of different overlap classification algorithms.

In Section 7, we describe a method to adapt parameters of overlap classification algorithms dynamically in real-time in order to maximize the system polygon throughput of the system. Finally, Section 8 contains a summary and conclusion.

The main contributions of this paper are: 1) the description and characterization of several different overlap classification algorithms, 2) the design of a prototype sort-first parallel rendering system utilizing a network of PCs, and 3) the introduction of the idea that it can be important to adjust parameters of an overlap classification algorithm dynamically to trade-off speed versus accuracy in order to achieve maximal system throughput in a parallel rendering system.

2 Previous Work

Classification of overlaps between 3D graphics primitives and 2D regions of the screen is a fundamental problem in computer graphics, arising in hidden surface removal [37], collision detection [15], occlusion culling [19, 22], and parallel rendering [4, 39].

In particular, partitioning primitives based on tile overlaps is an important step in many high-performance polygon rendering systems utilizing image-space parallelism or bucket rendering. Different tile shapes have been used in these systems, including scan lines [14], horizontal strips [3, 20, 38], vertical strips [38], and rectangular areas [1, 6, 20, 33, 40]. Tiles have been kept static [1, 6, 7, 20, 27, 31] or dynamically adjusted based on the distribution of graphics primitives on the screen [28, 33, 38]. For each graphics primitive, the rendering system must determine which 2D screen-space tiles it overlaps so that it can invoke rendering operations on the appropriate processors.

Most current parallel polygon rendering systems are based on a sort-middle architecture. Some hardware implementations rely upon a fast, global interconnection to distribute primitives from geometry processors to rasterizers. For instance, SGI's Infinite Reality Engine [27] uses a fast Vertex Bus to broadcast screen space vertex information from semi-custom ASIC geometry processors to all ASIC rasterization processors. In this case, overlaps between primitives and tiles are not determined by the geometry processors, and every vertex is broadcast to every rasterization processor, requiring each rasterization processor to check all primitives to see if they overlap its screen region. Most other sort-middle systems have used the 2D bounding box of a primitive to test for overlaps with screen-space tile regions. For example, this method is used in UNC's PixelPlanes 5 to sort primitives among tiles loaded onto a work queue to be processed by multiple rasterization processors [11]. Other examples include Renderman [36], PixelFlow [9, 25], and several commercial PC-based graphics accelerators [6].

Several studies have investigated the impact of primitive overlaps in bucket rendering systems. Molnar proposed an equation for modeling the overlap factor for 2D bounding boxes on 2D rectangular tiles [24]. His analytical model has been corroborated by experimental evidence [6, 26], and it has been used as the basis for subsequent studies [1, 6]. Cox and Bhandari investigated the relationships between tile (bucket) size and overlap factors incurred in a bucket rendering system [6]. In their experiments, they sort every primitive among the tiles according to the overlaps of its 2D bounding box constructed after projecting the primitive into screen space. Although they describe alternative overlap classification methods (e.g., exact bucket sorting), they do not examine their costs or the extent to which they may reduce overlap. Chen et al. studied the effect of overlap factors on rendering times in bucket rendering systems [1]. Based on analytical models and experimental evaluations, they concluded that the processing overhead due to overlaps is generally far less than the raw overlap factor. To our knowledge, there has not been previous work studying different overlap classification algorithms for bucket rendering systems.

Our investigation is based on an immediate-mode sort-first approach [26]. Sort-first has been studied for retained-mode systems by Mueller [28] and Cox [5], among others. Mueller simulated a real-time sort-first system, primarily studying load balancing [28] and dynamic distribution of hierarchical scene descriptions [29], while Cox [5] implemented a sort-first version of Renderman [36].

Relatively little work has been done on interactive polygon rendering using a cluster of networked PCs [17, 34]. Prior distributed rendering systems have mostly focused on inter-frame parallelism [17], rendering separate frames of a sequence on separate computers in parallel, rather than on intra-frame parallelism, which is required to achieve scalable speedups in a real-time system utilizing many processors. Networks of workstations have been used successfully for parallel graphics algorithms with coarse granularity, such as volume rendering [23, 16], radiosity [32, 13], and batch rendering of image sequences [17, 30]. We are not aware of any prior system that has achieved scalable polygon rendering speedups via intra-frame data parallelism across a network of workstations.

3 Problem Statement

In our investigation, we aim to construct a sort-first system matching the abstraction shown in Figure 1. Applications running on one or more clients generate streams of 3D graphics rendering commands (e.g., OpenGL) representing sequences of state changes, transformation operations, and 3D primitives. Each command is encoded and inserted into a bit stream to be sent via network messages from each client to the server(s) corresponding to the 2D tile regions potentially overlapped by the 3D primitives. Meanwhile, each server receives a set of messages, decodes them, and executes the corresponding 3D graphics rendering commands to create images for display. Parallelism is achieved by overlapping computation in the clients and servers (in a two-stage pipeline) and by allowing multiple servers to render graphics primitives covering different tiles of the screen concurrently.

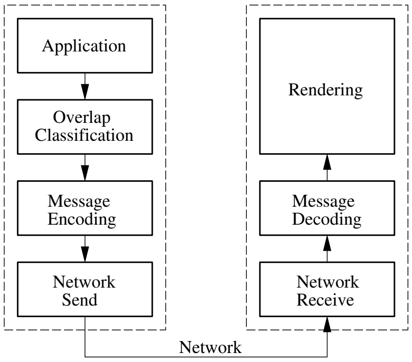


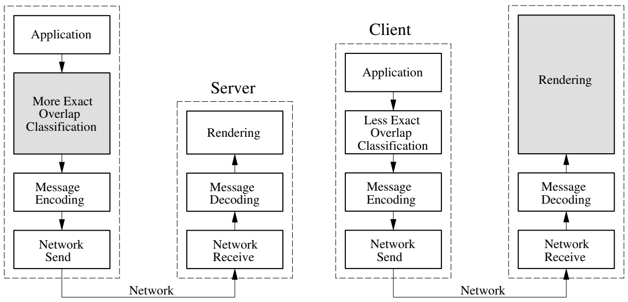
Figure 1: System abstraction.

An important challenge in this type of system is to develop client software that distributes 3D primitives among the servers efficiently. Of course, since each server is responsible for rendering the pixels in only one tile of the screen, it is generally not necessary for every server to receive and process every 3D graphics primitive. Rather, each server must render only the subset of 3D graphics primitives at least partially overlapping its tile. We can take advantage of this fact by checking which 3D primitives overlap which 2D tiles in the client prior to sending the primitives to the servers. If the client can determine that a particular 3D graphics primitive, P_i, does not overlap a particular tile, T_j, then it can omit P_i in the stream of commands to be processed by the server rendering the image for T_j. Although attribute state changes must still be processed by every server, wasteful geometry computations to process 3D primitives entirely outside a server's tile can be avoided. Since the geometry processing time required simply to clip every 3D primitive in every server is significant, incorporating such an overlap classification algorithm into the client is critical to building a scalable system supporting a large number of servers.

In our system abstraction, the overlap classification algorithm processes an "immediate-mode" stream of 3D graphics primitives and decides in real-time which tiles are potentially overlapped by each primitive. Since the overlap results are used by the client to determine which primitives to send to each server, the algorithm must be conservative (i.e., it should not miss any overlaps). Yet, it should be as quick and accurate as possible. Moreover, since we aim to support interactive applications, the classification algorithm should add as little latency to the rendering pipeline as possible, and the total latency of the rendering system should not exceed the tolerance of typical interactive graphics applications. This constraint may impose a limit on the number of primitives that can be considered by the algorithm before any of the primitives is classified. Accordingly, it is usually not feasible to perform precomputation to build complicated spatial data structures based on the locations of all the 3D graphics primitives in an image.

Clearly, there are trade-offs between spending more processing time and achieving more exact overlap classifications. On one extreme, the client could do no processing and simply send every primitive to every server (this approach is very quick, but very conservative!). In this case, every server would have to process every primitive, at least enough to decide it lies outside its region of the screen, and the system would not scale well to support large numbers of servers. On the other extreme, the client could render all primitives into a full-resolution item buffer, and read back the polygon IDs to determine exact tile overlaps. Then, each server would have to render only polygons that actually contribute to the image covering its tile. Generally, if an algorithm takes more time to compute more exact overlaps, the client becomes the bottleneck and limits overall rendering throughput (see Figure 2a). Yet, if the classification algorithm takes less time, but is more conservatively approximate, then more graphics primitives are sent to be rendered on servers whose tiles are not actually overlapped, and the servers' rendering time becomes the bottleneck (see Figure 2b).

The challenge is to balance the competing goals of speed and accuracy in the overlap determination algorithm to achieve a balance between client and server processing so that the system's rendering throughput is maximized.



(a) More exact overlaps. (b) More approximate overlaps.

Figure 2: More approximate overlap classifications shift processing burden from the client to the servers.

4 Overlap Classification Algorithms

We have developed several algorithms that classify 3D primitives based on their overlaps with 2D tiles. The various algorithms are based on different conservative approximations that trade-off speed versus accuracy. For instance, we consider different bounding volume approximations, different transformation sequences, and different primitive aggregation methods. The algorithms and their parameters are described in the following subsections, and then results of experiments with these algorithms are discussed in Section 6.

4.1 Bounding Volume Approximations

It is common in computer graphics to use a bounding volume as a surrogate for a primitive in conservative overlap and intersection computations [2]. Candidate bounding volumes include axis-aligned bounding boxes, bounding spheres, oriented bounding boxes, K-dops, convex hulls, and even the original 3D primitive, where each of these candidates represents a different point in the spectrum of trade-offs between accuracy and computational expense. Previous tiled rendering systems have used 2D axis-aligned bounding boxes to compute tile overlaps.

Determining overlaps based on primitive bounding volumes is intuitively appropriate because the computations are both conservative and relatively quick. In our system abstraction, the client must feed T servers with primitives. Therefore, in order to keep up with the servers, it must be able to determine the set of tiles overlapped by a primitive at least T times faster than the servers can draw them. Of course, it may not be practical to determine overlaps between every 3D primitive and the 2D tiles exactly. In essence, exact overlap classification requires tile-resolution rasterization of every 3D primitive in the client. In this paper, we consider computing tile overlaps both exactly and approximately using axis-aligned bounding boxes. We did not investigate other bounding volume shapes because we found in early studies that using simple axis-aligned bounding boxes resulted in overlaps quite similar to any more exact bounding volume.

4.2 Transformation Sequences

As noted earlier, 3D graphics primitives generally appear in the command stream of a sort-first system as a sequence of 3D vertices located in a 3D modeling coordinate system (e.g., as in OpenGL), while the tiles for which we wish to classify overlaps are described in the 2D screen-space coordinate system. As a result, we must transform one or both to a common coordinate system in order to check for overlaps. For example, in OpenGL, the transformation between these two coordinate systems is defined by the concatenation of the matrices stored at the tops of the "model view" and "projection" transformation stacks.

There are several possible coordinate systems in which primitive-tile overlap computations could be performed, including: 1) the 3D modeling coordinate system of the 3D primitive, 2) the 3D world coordinate system of the scene, 3) the 3D camera coordinate system, or 4) the 2D screen-space coordinate system [10]. For instance, it would be possible to check for overlaps in 3D modeling coordinates by computing the 3D view frustum corresponding to the region of space in camera coordinates mapping to each 2D tile of the screen, and then transforming that 3D view frustum by the inverses of the modeling and camera matrices to form a 3D volume in 3D modeling coordinates suitable for determining overlaps with each 3D primitive. Clearly, the efficiency of the overlap classification algorithm will depend greatly on the sequences of transformations applied to the primitives and/or tiles and the coordinate system used for overlap computations. Likewise, combining different transformation sequences with different bounding volume approximations yields interesting possibilities (some of which are shown in Figure 3).

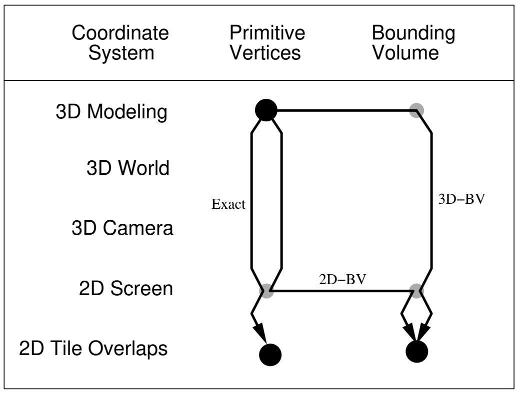


Figure 3: Possible transformation sequences.

In this paper, we consider the following four overlap classification algorithms: [1]

- **Exact:**
  1) Transform the 3D vertices of each 3D polygon to 2D screen-space, 2) Scan convert the projected 2D polygon to find tile overlaps exactly. (the left-most path in Figure 3).
- **2D-BV:**
  1) Transform the 3D vertices of each 3D polygon to 2D screen-space, 2) Compute a 2D bounding volume enclosing the projected vertices, 3) Check for overlaps between the 2D bounding volume and the 2D tile regions. (the middle path in Figure 3).
- **3D-BV:**
  1) Compute a 3D bounding volume enclosing the original 3D vertices, 2) Transform the 3D bounding volume to 2D screen-space, 3) Compute a 2D bounding volume enclosing the projected 3D bounding volume, 4) Check for overlaps between the 2D bounding volume and the 2D tile regions. (the right-most path in Figure 3).

[1] In this discussion, we focus on classification of overlaps for 3D polygons. Yet, the same principles apply to other 3D primitives as well.

- **All:**
  1) Trivially assume every 3D polygon overlaps every 2D tile region.

4.3 Primitive Aggregation Methods

There are likely to be cases where the client is not able to keep up with the servers if it computes the tile overlaps for each 3D graphics primitive independently. However, since applications usually generate streams of primitives with high spatial coherence (e.g., all the polygons for each object of the scene are drawn in sequence), it seems like a good idea to collect a number of primitives into one aggregate primitive and then perform overlap classification for this collection as a whole in a single step. Then, each overlap computation is amortized over multiple primitives, effectively reducing the classification overhead in the client.

We integrate this amortization strategy with the 2D-BV and 3D-BV algorithms described in the previous section. Specifically, we compute one bounding volume for a group of primitives, determine the tile overlaps for that bounding volume, and then assign every primitive in the group to all the tiles overlapped by the bounding volume. This method is rather conservative in that it assumes that every primitive in overlaps at least all the tiles overlapped by any primitive in the group.

The benefits of amortization are directly related to the amount of work avoided for each primitive. In the 3D-BV algorithm, all per primitive computations can be amortized except the inclusion of a primitive into the current 3D bounding volume (i.e., steps 2-4 can be amortized). In the 2D-BV algorithm, each primitive must be transformed before it is included in the current 2D bounding volume, and thus only the 2D computation to check the overlaps between the current bounding volume and tiles can be amortized (step 3). In either case, the algorithm becomes more conservative, yet runs faster, as we increase the amortization factor, A, representing the number of primitives loaded into each bounding volume, thereby allowing an interesting trade-off between speed and accuracy.

5 Experimental System

In order to investigate the trade-offs of the different overlap classification algorithms described in the previous section, and to study the feasibility of constructing a parallel rendering system utilizing a cluster of PCs, we built an experimental parallel rendering system using eight AccelGraphics Eclipse graphics accelerator cards [8] inside 200MHz PC-SMPs attached to a Myrinet network to drive an array of eight 1024x768 Proxima 9200 LCD projectors. The images rendered by the eight servers are projected on a rear-projection screen in a 4x2 grid to form a single image covering the entire screen (see Figure 4). The effective display resolution of this system is around 4000x1500 pixels, while the potential rendering performance is eight times that of a single PC. [Note to the reviewers: we expect to experiment with a system comprising 15 server PCs driving a 5x3 array of projectors before the conference].

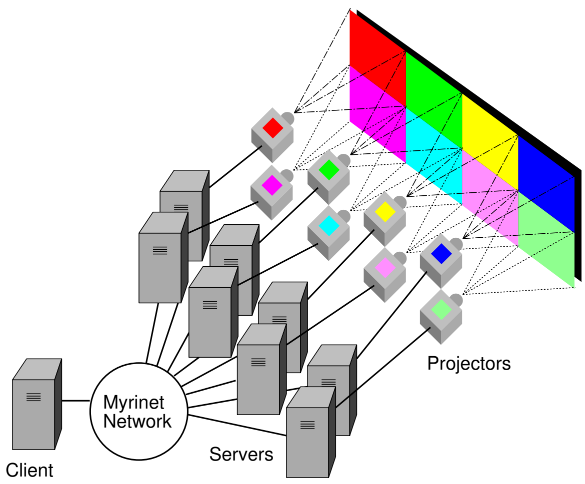


Figure 4: Experimental system organization.

A single client PC with a 300MHz Pentium II processor drives the system by running an OpenGL executable dynamically linked with a special OpenGL library (linked in Windows NT by simply placing the opengl.dll file in the local directory). Rather than invoke the OpenGL commands locally, this version of the OpenGL library constructs messages encoding the OpenGL function calls and arguments, runs one of the algorithms described in the previous section to insert into the message a bit vector representing the set of server tiles overlapped by each primitive, and sends the messages over the Myrinet network to the servers. Each server receives a set of messages, decodes them, checks the bit vectors, and conditionally executes the decoded OpenGL functions to create images for its projector. If the system is executing in double buffer mode, the client and server exchange synchronization messages at the end of each frame so that all the servers swap buffers at approximately the same time.

The sort-first approach is well-suited for our prototype system because it takes advantage of the natural screen-space parallelism inherent in our cluster of server PCs. Also, since the 4x2 grid of tiles assigned to servers exactly matches the 4x2 grid of projection regions on the screen, no extra processing or network communication is required to composite subimages for the separate tiles into a final full-screen image for display. Instead, subimages are composited "optically" during projection since the projected images blend as light from different projectors combines on the screen. A problem with our current system is load imbalance. In related, but separate, work, we are investigating algorithms that dynamically adjust tile shapes and sizes to balance the rendering load among the servers, and we are investigating methods for efficient inter-server communication methods to distribute rendered pixels for display at the end of each frame.

6 Results

We executed a series of experiments with this prototype system using two OpenGL applications. The first application is an unmodified VRML browser. It reads a 3D model from a file and displays it on the screen while the viewpoint is moved either interactively or along a predetermined path. We executed tests with this application using the following three test models with predetermined camera paths captured apriori during an interactive session:

- **Eagle:** Exterior view of finely tesselated object (15,154 triangles).
- **Building:** Interior view of hierarchical scene graph with large surfaces (29,974 quadrilaterals).
- **Suite:** Interior view of hierarchical scene graph with both large surfaces and finely tesselated objects (80,372 polygons).

The second application, which we call "Triangles," generates "randomized" front-facing right-triangles with a given 2D projected area uniformly distributed across the screen. The triangles are created by the following algorithm which is iteratively given a 2D point, P, on the screen and a desired screen-space area, A. First, construct a 2D vector V with a random orientation and a random length. Next, assign one edge of the triangle as the line segment leaving from point P along V. Then, choose the remaining two edges such that one of them leaves P in a direction perpendicular to V such that the resulting triangle has area A. During our tests, we generated 10,000 triangles with this method, iteratively perturbing the point P from triangle to triangle. Parameters to the triangle application control: 1) the number of triangles to be generated, and 2) the 2D projected area of each generated triangle.

6.1 Bounding Volume Approximation & Transformation Sequence Results

In our first experiment, we investigated the effectiveness of the four different overlap classification algorithms described in Section 4.2. In this experiment, tile overlaps were computed in the client for every primitive individually (i.e., the amortization factor was 1).

We ran two sets of tests. The first test used the Triangles application to generate 10,000 "random" triangles covering 2D projected areas of 250, 500, 1000, 2000, 4000, and 8000 pixels, respectively. The second test used the VRML browser to view the three 3D test models. The results comparing the speed and accuracy of the different algorithms measured in these tests are plotted in Figure 5. In each plot, the horizontal axis (labeled "Overlap Computation Time Per Primitive") represents the speed – it is the wall-clock time (in microseconds) required by the client to classify potential overlaps for each polygons in a test. Meanwhile, the vertical axis (labeled "Average Computed Overlap Factor") represents the accuracy – it is the number of 2D tiles determined to be potentially overlapped by each 3D primitive on average.

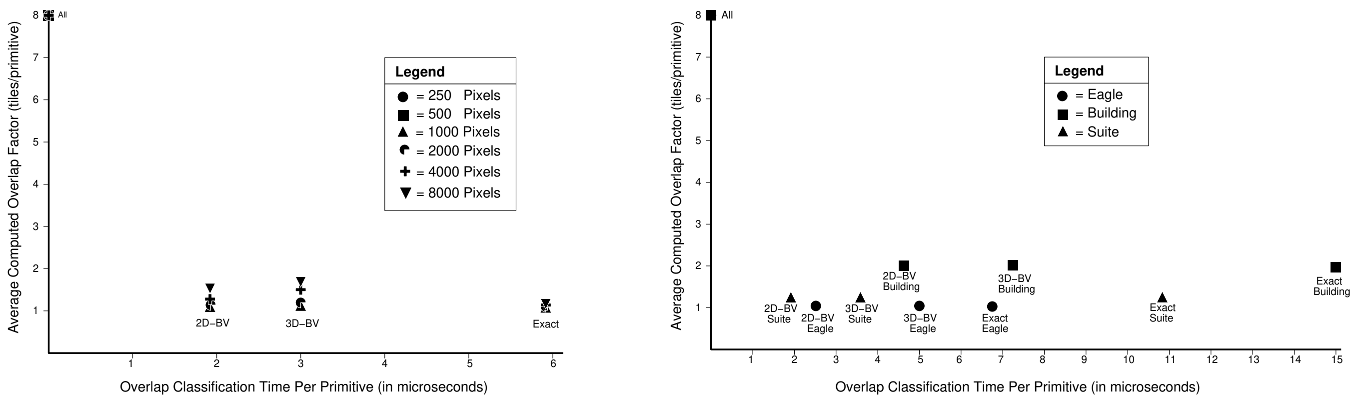
There are several interesting results evident in these plots. First, the client is not able to classify overlaps for every primitive individually and still achieve polygon throughput rates matching high-performance rendering systems. The fastest of the non-trivial overlap algorithms requires 1.9us to process each primitive individually on a 300 MHz Pentium II processor, yielding a sorting rate of 526K polygons/second. Exact overlap classification for the triangle tests requires 5.9us per polygon (or 169K polygons/second), while exact overlaps in the VRML browser test took as much as 14.8us per polygon (or 68K polygons/second). The increased time for overlap classification in the tests with the VRML 'Building' model are due primarily to the costs of numerous transformation operations included in the hierarchical scene description.

Second, the 3D-BV algorithm is significantly slower than the 2D-BF algorithm. This result is primarily due to the fact that 3D-BV must transform eight vertices from 3D to 2D for each triangle (i.e., the corners of the 3D bounding box), while the 2D-BV algorithm only must transform the three vertices of the triangle. The matrix multiplications performed for each vertex transformation is the limiting factor in these algorithms, and thus we expect that the 3D-BV algorithm would take less time than the 2D-BV algorithm for polygons containing more than eight vertices and for other higher-level primitives.

Third, we note that the bounding volume approximations have little impact on the overlap factor in cases where the polygons are significantly smaller than the tile size. As the polygons approach a size approximately 1/200th of the tile size, the differences between exact overlap classification and approximations based on the 2D-BV and 3D-BV algorithms start to become significant.

6.2 Amortization Results

In our second experiment, we investigated the trade-offs of increasing the amortization factor for the 2D-BV and 3D-BV algorithms. We ran two sets of tests for each of the algorithms with varying amortization factors (A). First, as before, we used the Triangles application to construct 10,000 "random" triangles covering 2D projected areas of 250 and 8000 pixels, respectively. Second, we used the VRML browser application to view the three 3D test models. Plots comparing



(a) Triangle tests. (b) VRML tests.

Figure 5: Comparison of speed and accuracy for four different overlap classification algorithms tested with data sets comprising (a) triangles of different areas, (b) three VRML models.

the measured speed and accuracy of the 2D-BV and 3D-BV algorithms with different amortization factors are shown in Figure 6. The labels and meanings of the axes are the same as in Figure 5.

From the results of this experiment, we see that it is possible to trade-off speed for accuracy by adjusting the amortization factor for either the 2D-BV or 3D-BV algorithms. Although the benefits of increasing the amortization factor diminish as it gets larger (due to per primitive processing overheads), we are able to improve the overlap processing times in the client significantly with amortization (usually by more than four times). In the fastest case, the client spends only 0.7us classifying each polygon (1.4M polygons/second). But, in spending less time, it performs more approximate checks, and the overlap factor increases, resulting in higher rendering times in the servers. The benefits/costs of larger amortization factors will depend on many factors, including the system communication rates, server rendering rates, etc.

We find that there is no single value for the amortization factor that is best in all circumstances. In particular, as rasterization time in the server increases due to more complex lighting, larger triangles, mip-map textures, etc., the time required to classify overlaps for each primitive remains virtually the same. As a result, for each set of rendering attributes, the amortization factor at which the client processing and server rendering times are balanced is different. As an example, Table 1 shows results collected during an experiment with the 3D-BV algorithm tested with the Triangles application for triangles with a different 2D projected area. The first column of the table contains the 2D projected triangle area, while the second column lists the amortization factor used in the test. The third column shows the average computed overlap factor, and the third and fourth columns contain the wall-clock times required by the client to classify overlaps and pack primitives into messages (which usually takes around 8us per primitive), and by the server to render the primitives, respectively. Examining the rightmost two columns, it is easy to see that the amortization value at which the client and server times are balanced varies widely with different triangle sizes (the cross-over amortization values are highlighted in bold). Specifically, an amortization value of 1 provides the best system throughput in the tests with very large triangles (e.g., 8,000 pixels), while an amortization value between 16 and 32 balances the client and server times best in the test with small triangles (e.g., 250 pixels). Quite simply, there is no single amortization value that provides the best performance for all 3D models.

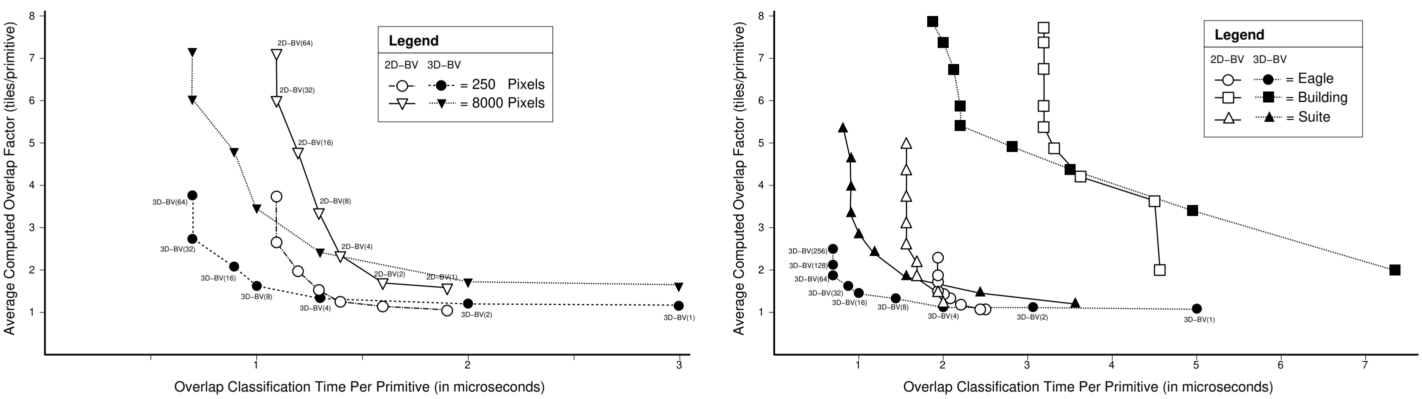
Finally, we observe that server rendering times (shown in the right-most column of Table 1) increase significantly (by more than a factor of 2) with more conservative overlap classifications. From this observation, we conclude that the geometry processing required by each server to clip 3D primitives projecting to areas entirely outside its tile is significant, and thus calculating reasonably exact tile overlaps is important to constructing a scalable sort-first rendering system.

7 Dynamic Amortization

Based on the results in the previous section, we are motivated to experiment with predictive feedback algorithms to adjust the amortization value, A, in real-time while processing a stream of graphics primitives. Our goal is to develop dynamic overlap classification algorithms that adjust the amortization value used in the 2D-BV and 3D-BV algorithms adaptively in order to balance client and server processing times so that the overall system throughput is maximized.

If we ignore other processing or communication overheads in the client (this is an idealized assumption!), we can model the client processing time per primitive, C, as:

$$C = C_P + C_{BV}/A \tag{1}$$



(a) Triangle tests.

(b) VRML tests.

Figure 6: Comparison of speed and accuracy for different amortization factors using the 2D-BV and 3D-BV algorithms tested with data sets comprising (a) triangles of different areas, (b) three VRML models.

where C_P and C_{BV} are the predicted processing times required by the client for each primitive and bounding volume, respectively, and A is the amortization factor.

We can also predict the rendering time required by the server, T_S, as:

$$S = max(OS_P, hwS_R/A) \quad (2)$$

where O is the predicted overlap factor, w and h are the predicted width and height of primitives to be rendered, and S_P and S_R are the predicted rendering times required by the server for each primitive and pixel, respectively.

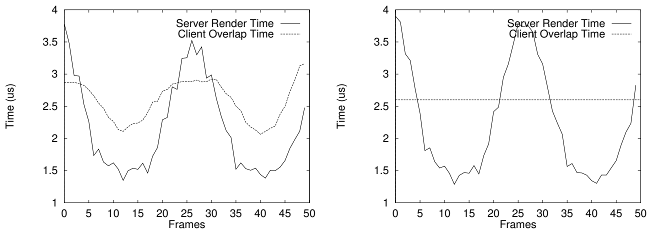
We can model a system in which client and server processing times are balanced, and thus the polygon throughput is maximized, by equating C and S, for example:

$$C_P + C_{BV}/A = max(OS_P, hwS_R/A) \quad (3)$$

Of course, this equation contains many variables unknown for the upcoming set of 3D primitives in an immediate-mode graphics system. Although we can determine reasonable estimates for the client processing time parameters, C_P and C_{BV}, and the server rendering time parameters, S_P and S_R empirically [12] (at least for a given rendering mode), we do not know the overlap factor, O, nor the projected screen-space width and height of future 3D primitives apriori. To circumvent this problem, we assume that the 3D primitives sent by an application have a large amount of coherence in their primitive sizes, and we predict values for future frames based on values for recently rendered primitives. Then, we solve the following equation for A_i to update the amortization factor, each time a new bounding volume is processed by the 2D-BV or 3D-BV algorithms:

$$C_P + C_{BV}/A_i = max(O_{i-1}S_P, h_{i-1}w_{i-1}S_R/A_{i-1}) \quad (4)$$

With this enhancement, we are able to achieve more balanced processing between client and server. Figure 7a shows the client overlap classification times (using 3D-BV) and server rendering times per polygon measured during execution of a modified version of our triangle generation program in which the length, L, used to construct triangles was modulated up and down with a cosine function to produce triangles with areas between 500 pixels and 1500 pixels. Figure 7b shows similar times measured without dynamically updating the amortization factor (using A=4). The client and server times track each other more closely using the dynamic algorithm, resulting in higher system throughput.



(a) Dynamic amortization.

(b) Static amortization (A=4).

Figure 7: Measured client and server processing times for each frame using overlap classification algorithms with (a) dynamic and (b) static amortization.

8 Conclusion

We have described algorithms for sorting 3D primitives among 2D screen-space tiles and investigated their use in an immediate-mode, sort-first parallel rendering system utilizing a network of commodity PCs. Based on experiences, we draw the following conclusions:

- Calculating tile overlaps is important to constructing a scalable sort-first rendering system.

Triangle Area	Amort Factor	Overlap Factor	Client Time	Server Time
250	1	1.09	3.8	0.9
	2	1.22	2.8	1.1
	4	1.39	2.1	1.3
	8	1.63	1.8	1.3
	16	**2.05**	**1.7**	**1.5**
	32	**2.71**	**1.5**	**1.6**
	64	3.78	1.5	1.9
500	1	1.09	3.8	1.1
	2	1.25	2.8	1.3
	4	1.46	2.1	1.5
	8	1.76	1.8	1.6
	16	**2.27**	**1.7**	**1.7**
	32	**2.98**	**1.5**	**1.9**
	64	4.12	1.5	2.2
1,000	1	1.09	3.8	1.2
	2	1.30	2.8	1.6
	4	**1.55**	**2.1**	**1.6**
	8	**1.95**	**1.8**	**1.9**
	16	2.60	1.7	2.0
	32	3.51	1.5	2.3
	64	4.82	1.5	2.6
2,000	1	1.12	3.8	1.7
	2	**1.38**	**2.8**	**2.1**
	4	**1.72**	**2.1**	**2.5**
	8	2.25	1.8	2.7
	16	3.12	1.7	3.0
	32	4.29	1.5	3.2
	64	5.59	1.5	3.5
4,000	**1**	**1.49**	**3.8**	**3.2**
	2	**1.51**	**2.8**	**4.2**
	4	2.01	2.1	4.7
	8	2.77	1.8	5.0
	16	3.91	1.7	5.3
	32	5.18	1.5	5.5
	64	6.36	1.5	5.6
8,000	**1**	**1.20**	**3.8**	**9.3**
	2	1.70	2.8	9.5
	4	2.40	2.1	9.9
	8	3.47	1.8	10.1
	16	4.79	1.7	10.3
	32	6.04	1.5	10.3
	64	7.11	1.5	10.4

Table 1: Statistics gathered with triangle sizes and different amortization factors using the 3D-BV algorithm (times are per primitive in microseconds).

- It is possible to trade-off speed and accuracy with different overlap classification algorithms.
- Our fastest conservative algorithm (3D-BV) can process approximately one million triangles per second while generating overlap factors around twice the ones computed with the exact algorithm.
- Dynamic adjustment of the speed/accuracy of the overlap classification can improve overall system throughput.

We have addressed just a few of the many issues related to building a sort-first parallel rendering system using a network of PCs. Other issues for future work include: 1) efficient network communication, 2) load balancing, and 3) image composition. In particular, networking bandwidth limitations present very significant challenges. In our prototype system, polygon throughput is limited by the capacity of the network connecting the client to the servers (40MB/s), and thus we are able to achieve only 80K polygons/second in typical OpenGL programs. Of course, this problem of delivering data to fast rendering hardware is classical in the design of computer graphics systems. Future work should be directed at improving the bandwidths available on clusters of PCs and investigating high-level graphics primitives and parallel APIs (e.g., [18]) to overcome communication bandwidth limitations.

Overall, we are encouraged by the results of our initial investigations into sort-first rendering systems utilizing a network of PCs, and we are hopeful that it will be possible to scale this approach to produce high-performance, high-resolution computer graphics systems in the future.

References

[1] Milton Chen, Gordon Stoll, Homan Igehy, Kekoa Proudfoot, and Pat Hanrahan, Simple Models of the Impact of Overlap in Bucket Rendering, *1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware,* Lisbon, Portugal, 1998, 105-112.

[2] Jim Clark, Hierarchical Geometric Models for Visible Surface Algorithms, *Communications of the ACM*, 19, 10, October, 1976, 547-554.

[3] T.W. Crockett and T. Orloff, A MIMD Rendering Algorithm for Distributed Memory Architectures, *Proc. Parallel Rendering Symposium*, ACM Press, New York, Oct. 1993, 35-42.

[4] T.W. Crockett, Parallel Rendering, In *Encyclopedia of Computer Science and Technology,* Vol. 34, Supp. 19, A. Kent and J. G. Williams, eds., Marcel Dekker, 1996, pp. 335-371. Also available as ICASE Report No. 95-31 (NASA CR-195080), April 1995.

[5] Michael Cox, *Algorithms for Parallel Rendering*, Ph.D. thesis, Department of Computer Science, Princeton University, May, 1995.

[6] Michael Cox, Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC, *1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware,* Los Angeles, CA, 1997, 25-34.

[7] D. Ellsworth, A New Algorithm for Interactive Graphics on Multicomputers, *IEEE Computer Graphics and Applications*, Vol 14, No. 4, July 1994, 33-40.

[8] Evans and Sutherland, *REALimage(TM) 1000 Hardware Manual*, Evans&Sutherland, 1997.

[9] J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, and L. Westover, PixelFlow: The Realization, *Proceedings of the 1997 Siggraph/Eurographics Workshop on Graphics Hardware*, Los Angeles, CA, Aug. 3-4, 1997. Pages 57-68.

[10] Foley, J.D., A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice.* 2nd ed., Addison-Wesley, Reading, MA, 1990.

[11] H. Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebss, and Laura Israel, Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories, *Computer Graphics* (SIGGRAPH 89), 23, 3, July 1989, 79-88.

[12] Funkhouser, Thomas A., and Carlo H. Séquin. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. *Computer Graphics (SIGGRAPH '93)*, 27, 247-254..

[13] T. Funkhouser, Coarse-Grained Parallelism for Hierarchical Radiosity Using Group Iterative Methods, *Computer Graphics* (SIGGRAPH 96), August 1996.

[14] Peter N. Glaskowsky, Advanced 3D chips show promise, *Microprocessor Report*, 11, 8, June, 1997, 5-9.

[15] Stefan Gottschalk, Ming Lin, Dinesh Manocha, OOB-Tree: A Hierarchical Structure for Rapid Interference Detection, *Computer Graphics* (SIGGRAPH 96), August 1996.

[16] C. Grietsen and J. Petersen, Parallel Volume Rendering on a Network of Workstations, *IEEE Computer Graphics and Applications*, 13, 6, November 1993, 16-23.

[17] Jeremy Hubbell, Network Rendering, *Autodesk University Sourcebook*, Vol. 2, Miller Freeman, 1996, 443-453.

[18] Igehy, Homan, Gordon Stoll, and Pat Hanrahan, The Design of a Parallel Graphics Interface, *Computer Graphics* (SIGGRAPH 96), July 1998, 141-150.

[19] Jones, C.B. A New Approach to the 'Hidden Line' Problem. *The Computer Journal*, 14, 3 (August 1971), 232-237.

[20] M. Kaplan and D.P. Greenburg, Parallel Processing Techniques for Hidden Surface Removal, *Computer Graphics* (SIGGRAPH 79), 13, 2, July, 1979, 300-307.

[21] Renate Kempf, Chris Frazier, *OpenGL Reference Manual*, 2nd edition, Addison-Wesley, 1992

[22] Luebke, David P., and Chris Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. *Computer Graphics*, Special Issue on 1995 Symposium on Interactive 3D Graphics, Monterey, CA, 1995.

[23] K.L. Ma, J.S. Painter, C.D. Hansen, and M.F. Krogh, Parallel Volume Rendering Using Binary-Swap Compositing, *IEEE Computer Graphics and Applications*, 14, 4, July 1994, 59-68.

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

[25] S. Molnar, J. Eyles, and J. Poulton. PixelFlow: High-Speed Rendering Using Image Composition, *Computer Graphics* (SIGGRAPH 92), July 1992, 231-240.

[26] 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, 23-32.

[27] J.S. Montrym, D.R. Baum, D.L. Dignam, and C.J. Migdal, InfiniteReality: A Real-Time Graphics System, *Computer Graphics* (SIGGRAPH 97), August 1997, 293-303.

[28] Carl Mueller, The Sort-First Rendering Architecture for High-Performance Graphics, *Computer Graphics*, ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3-D Graphics, April 1995.

[29] Carl Mueller, Hierarchical Graphics Databases in Sort-First, *Proceedings of the IEEE Symposium on Parallel Rendering*, 1997, 49-57

[30] Pixar, *PhotoRealistic RenderMan Toolkit*, 1998.

[31] F. I. Parke, Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems, *Computer Graphics* (SIGGRAPH 80), 14, 3, July 1980, 48-56.

[32] R.J. Recker, D.W. George, and D.P. Greenberg, Acceleration Techniques of Progressive Refinement Radiosity, *Computer Graphics* (Proceedings of the 1990 Symposium on Interactive 3D Graphics), 24, 2, March 1990, 59-66.

[33] D.R. Roble, A Load Balanced Parallel Scanline Z-Buffer Algorithm for the iPSC Hypercube, *Proc. Pixim 88*, Hermes, Paris, France, October, 1988, 177-192.

[34] Bengt-Olaf Schneider, Parallel Rendering on PC Workstations, *International Conference on Parallel and Distributed Processing Techniques and Applications* (PDTA98), Las Vegas, NV, 1998.

[35] J. Torborg and J. Kajiya, Talisman: Commodity Realtime 3D Graphics for the PC, *Computer Graphics* (SIGGRAPH 96), August 1996.

[36] S. Upstill, *The Renderman Companion*, Addison-Wesley, Reading, MA, 1989.

[37] J. Warnock, A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures, Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake City, June, 1969.

[38] D.S. Whelan, Animac: A Multiprocessor *Architecture for Real-time Computer Animation*, Ph.D. dissertation, California Institute of Technology, Pasadena, CA, 1985.

[39] S. Whitman, Multiprocessor Methods for *Computer Graphics Rendering*, A.K. Peters, Wellesley, MA, 1992.

[40] S. Whitman, Dynamic Load Balancing for Parallel Polygon Rendering, *IEEE Computer Graphics and Applications*, Vol 14, No. 4, July, 1994, 41-48.