High-Speed Rendering using Scan-Line Image Composition

A Dissertation Proposal by
Steven Molnar

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

Thesis

Scan-line rendering can be combined with real-time image composition to produce a flexible, scalable image-generation architecture with higher performance than existing architectures.

Abstract

This proposal describes a new architectural approach for high-speed rendering based on image-composition and scan-line rendering. The graphics database (or application producing graphics primitives) is partitioned and distributed over a homogeneous array of processors. Each processor transforms and rasterizes its respective primitives in scan-line order, computing subpixel information where appropriate. The pixel streams produced by the scan-line renderers are fed into an image-composition structure that combines them into a single pixel stream that describes the entire image. This pixel stream can either be stored in a conventional frame buffer, or, if the composition circuitry is sufficiently fast and if the flow of data is sufficiently uniform, it can drive a raster display directly. Such a system offers potential advantages of scalability over a wide performance range, low latency, and flexibility for many alternative input streams, such as volume data, and transparent surfaces. The object of this proposed research is to demonstrate that these properties can be achieved in a realizable system.

1. Introduction

A major thrust of research in computer graphics is developing hardware architectures to display graphics images rapidly, realistically, and economically. The field has progressed from monochrome systems that displayed only a few lines in real time in the 1960s, to raster systems that display up to one million realistically-shaded polygons per second in 1990 [SGI90]. Over this entire period, users have consistently demanded higher rendering speed, the ability to display more realistic images, and to do so at lower cost.

The display process in a raster computer image-generation (CIG) system is composed of two conceptual pipelined parts: geometric processing (frequently called the "front-end" subsystem), and rasterization (the "back-end" subsystem). Geometric processing encompasses the stages that transform the user's display model into a series of primitives in screen coordinates. Rasterization then converts these transformed primitives into pixels on a raster display screen. Figure 1 shows the conceptual pipeline for a typical, modern CIG system.

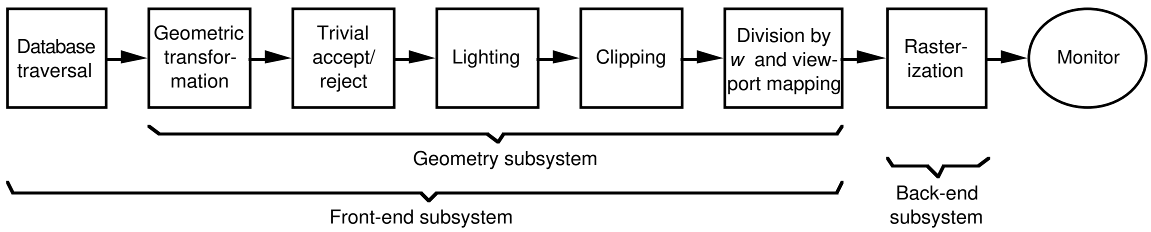

Figure 1. Conceptual pipeline for displaying Phong-shaded polygons.

Both geometric processing and rasterization are related; solutions to one problem inevitably affect the other. Both are computationally demanding in high-performance CIG systems as well, typically requiring extensive pipelining or parallelism to achieve desired rendering speeds. This proposal is largely concerned with rasterization, though it has ramifications for the front-end as well.

Parallel rasterization systems. Two traditional approaches for high-performance rasterization systems are image-parallelism (computing multiple pixels in parallel) and object-parallelism (processing multiple objects/primitives in parallel) [FOLE90].

In an image-parallel system, the frame buffer is divided into several partitions, and each partition is provided with its own rasterization processor, thereby increasing pixel processing and bandwidth into memory. This is the most common approach in commercial high-speed graphics systems. Its chief advantages are flexibility and high performance vs. price for moderate-performance systems. The performance of such architectures does not scale indefinitely, however; as more and more partitions are added, the percentage of time that each partition is doing useful work decreases. Indeed current image-generation architectures are rapidly approaching fundamental performance limits [GHAR89].

In an object-parallel system, a series of parallel *object processors* are each assigned one or more graphics primitives. The image is then traversed pixel-by-pixel in some fixed order (generally scan-line), with each object processor computing the contribution of its primitive(s) to the current pixel. The contributions from each object-processor are then prioritized in some manner (generally by comparing their z values) to determine which surface is visible. Object-parallelism in theory allows scalability beyond current performance levels, but the most-commonly-described variation, systems that allocate a processor for each primitive, restricts the class of primitives and rendering methods that can be used. A number of object-parallel systems have been proposed, though relatively few have actually been built.

Scan-line image composition. *Scan-line image composition* is a hybrid-parallel approach that combines many of the advantages of image and object-parallelism. Like other object-parallel systems, it is composed of a parallel array of identical processors. Rather than assigning one processor per primitive, however, each processor is assigned a number of primitives (typically several hundred or thousand). In other words, the entire graphics database is distributed over these processors without regard to final screen position. Each processor transforms and rasterizes its respective primitives in scan-line order. The pixel streams produced by the scan-line renderers are fed into an image-composition structure that combines them into a single pixel stream that describes the entire image. If the scan-line processors and the image-composition circuitry is sufficiently fast and the data flow is sufficiently predictable, the composite pixel stream can be displayed on a raster screen directly. More likely, it will be stored in a conventional frame buffer, and refreshed to the screen in the usual manner.

If image-composition can be made sufficiently efficient and "z-buffer" aliasing can be reduced to an acceptable level, such an architecture offers the following appealing properties:

- Modularity and expandability that allows systems to be built with arbitrarily high performance at a near-constant performance/price ratio.
- Image-generation latency approaching one frame time—the minimum possible (many high-speed graphics systems, particularly systems requiring a screen sort, have latencies of 2–3 frame times).
- Flexibility to render primitives other than opaque polygons or surfaces (such as volume datasets and transparent surfaces).

The object of this proposed research is to demonstrate that these properties can be achieved in a realizable system.

2. Related Work

This section describes previous work related to the proposed research.

Scan-line renderers. Scan-line rendering algorithms have been used in hardware and software graphics systems since the early days of computer graphics [WYLI67; BOUK70a; BOUK70b; WATK70]. Watkins' original scan-line system, built at the University of Utah in 1969-70, generated a video signal in real time for approximately 1200 polygons [WATK70]. This system achieved its performance using relatively few transistors—a necessity since integrated circuits were only beginning to become available. Evans and Sutherland developed several generations of real-time flight simulators using scan-line algorithms [SCHA83]. Scan-line rendering has been widely used in software rendering systems where high quality images are desired [WHIT82][CARP84].

Image Composition. Image composition has been used for many years as well. Simple overlays have been used to compose images in video editors, video games, and certain flight simulators—applications in which one image has absolute priority over another image. Video editors typically use chroma-keying, in which pixels in the primary image having a particular color (such as dark blue) are overwritten by pixels in the secondary image. A technique used in some flight simulators is to build a pipeline of processors, each of which generates an image of increasing priority. Pixels generated by one processor overwrite those generated by previous processors. This is effective, for example, when aircraft always obscure lights, which in turn always obscure surface terrain.

The full image-composition problem involves composing two images with pixels of different priorities. The simplest technique, used in most object-parallel systems, is to compare z-values to determine which pixel has priority. This composition method suffers from the same types of image aliasing as the conventional z-buffer algorithm. Duff [DUFF85] developed a composition algorithm that uses z-values at the four corners of each pixel to approximate the fractional contribution from each image. Although economical, this technique does not handle every case correctly (it is adequate for compositing uncorrelated images from a rendering toolkit—the purpose for which it was designed—but fails when primitives from different images share common edges). Nakamae et. al. described a technique based on decomposing multiple images into visible spans on scanlines with higher y resolution than the raster display [NAKA89]. The composite image is computed by merging spans from multiple images. The subscanline spans allow an antialiased color to be calculated for each pixel.

Processor-per-primitive systems. Several processor-per-primitive architectures have been proposed. The earliest, perhaps, was General Electric's NASA II [BUNK89]. This system allocated one polygon (face) per processor (face card) and rendered in scan-line order at video rates. Priorities between the faces were determined by a global priority algorithm. Cohen and Demetrescu proposed the first processor-per-primitive system in which surface priorities were decided using z-values and not global priorities [DEME80]. Pixels would stream through a pipeline of triangle processors with the color value of the closest surface emerging from the end. Fussell proposed a variation of this, in which a binary tree of comparators is used instead of Demetrescu's pipeline [FUSS82]. Weinberg proposed an enhancement to Demetrescu's system for antialiasing, in which variable-length packets containing subpixel information would be sent between pipelined polygon processors, and a filtering module would filter the subpixels appropriately [WEIN81]. Deering et. al. [DEER88] and Schneider [SCHN88] both proposed triangle-processor-pipeline systems in which shading is deferred to a final shading processor.

Image-composition systems. Several image-composition systems with multi-primitive renderers have been proposed as well. Bender et. al. proposed an image-composition system based on Inmos Transputers that uses a BSP-tree algorithm to compute polygon priorities and a DMA engine to combine pixels from several frame buffers into a composite frame buffer [BEND89]. Molnar described an architecture that combined the outputs of a number of z-buffer renderers using a binary tree of z-comparator/multiplexers and could achieve linearly scalable performance vs. price [MOLN88]. Shaw et. al. proposed implementing a simplified version of Duff's algorithm in VLSI chips to make possible a multi-renderer system with antialiasing [SHAW88]. This system is currently under construction at the University of Alberta.

Contribution of the proposed work. This proposal uses many ideas developed in the above work. It is closely related to other image-composition systems and some processor-per-primitive systems (particularly Weinberg's) but differs in several important respects:

- It is the first attempt to combine a large-scale scan-line renderer with image-composition. This reduces the need for intermediate storage, a major expense in other proposed systems.
- It seeks to address the antialiasing problem comprehensively. Shaw's system addresses antialiasing, but suffers from a serious difficulty: Duff's image-composition algorithm allows the color of background surfaces to "leak through" the seams between two adjacent polygons that form part of the same surface. For a scalable system, adjacent polygons must be able to reside on different renderers (in fact, this may be a necessary condition to assure even load balancing between renderers).
- It is intended to support a variety of rendering algorithms on different types of primitives, rather than being tailored to a single algorithm and primitive type.
- It addresses the problem of image-generation latency, an emerging problem in high-performance CIG systems.

The contribution of this research is to explore this new, potentially promising, architectural space and to demonstrate that many of its potential benefits can be realized.

3. A Prototype System

There's a long way between an idealized, high-level conception of an architecture and a feasible realization. It may not be possible, in fact, to achieve all of the properties listed above in a single system. This section describes a prototype scan-line image-composition system that could conceivably meet these goals. The individual parts of the system may or may not be feasible as described. The purpose here is to demonstrate a plausible outline of a system that could achieve the above properties and an avenue for initial research.

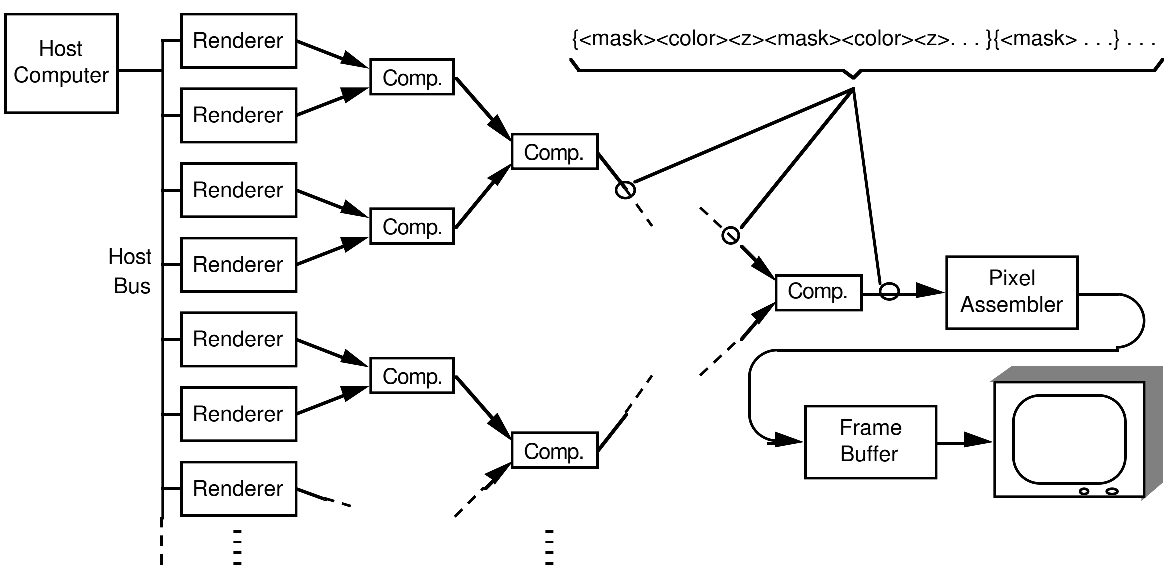

Figure 2: Prototype scan-line image-composition system.

The prototype system is composed of five main components, diagramed in Figure 2:

- A host computer
- A homogeneous array of parallel scan-line renderers
- A binary tree of compositors
- A pixel assembler
- A conventional frame-buffer and video display.

The host computer is a high-speed workstation with a disk drive, ethernet port, I/O devices, and other features necessary to support the user interface and I/O requirements of the graphics application. It need not have sufficient power to traverse, transform, or display the graphics database, as these functions are performed in other parts of the system.

The renderers are homogeneous parallel processors with sufficient memory to store a fraction of the graphics database, floating-point hardware to transform their share of the primitives, and (perhaps) special hardware for high-speed scan-line rasterization. Rather than computing a stream of RGB pixel values in scan-line order, the rasterizer here generates a stream of unassembled pixels. An unassembled pixel is an arbitrarily long packet of surface descriptors of the form:

<subpixel coverage bitmask> <surface color> <surface z-value>

Surface descriptors describe each surface that is potentially visible at a pixel. They are sorted in z-value from near-to-far. The list may be truncated when the first n surface descriptors completely cover the pixel (determined by *and* ing the subpixel coverage bitmasks).

Compositors are special-purpose devices that combine two streams of unassembled pixels into a single stream. Each of the inputs to the compositor contains a short FIFO queue to buffer incoming pixel streams. A custom compositor chip rapidly interleaves the incoming unassembled pixel streams according to z-values, and truncates the streams when the pixel has been fully covered. The stream of unassembled pixels leaving a compositor has the same form as the two incoming streams. A binary tree of compositors, therefore, combines the n streams from the renderers into a single stream of unassembled pixels emerging from the root of the tree.

The pixel assembler *assembles* or converts the stream of unassembled pixels into a stream of RGB color values. The pixel assembler computes the weighted sum of surface components based on the subpixel coverage bitmasks. Each position in the subpixel mask is assigned a coefficient based on an appropriate filter kernel. Each surface's contribution to the final RGB value for the pixel is based on the sum of its subpixel contributions.

The stream of RGB pixel values emerging from the pixel assembler is then stored in a double-buffered frame buffer, which allows screen refreshing at any rate desired, independent of the image-generation process.

Antialiasing. An important requirement for any graphics system, real-time, or otherwise, is that it display images without noticeable aliasing. Aliasing results from both the scan-conversion and visibility-determination stages of the rasterization process. Unfortunately, the obvious method for computing and combining digital images—using a single point-sample at the center of each pixel (the heart of the z-buffer algorithm)—is notorious for generating aliased images. Solving this problem requires scan-converting and determining visibility at a sub-pixel level of detail.

The two main approaches toward this are object-precision and image-precision antialiasing. In the object-precision approach, the boundaries and z-values of primitives are described geometrically, and the resulting coverage fraction is evaluated analytically. The difficulty with this approach is that many special cases are possible, and subsequent filtering to obtain a single pixel value is difficult.

The more popular approach is image-precision antialiasing. In this approach, the image is point-sampled at a higher level of detail where necessary. Although this technique can still miss details if image components are extremely small (small image features may slip between successive sample points), subpixel samples can be treated uniformly. A common variant of this technique is the A-buffer [CARP84], which we propose to adapt for our system. In the A-buffer approach, each pixel is represented by an arbitrarily long list of surface descriptors, each containing a subpixel coverage bitmask, a color value, and a z-value. Note that subpixel samples need not be distributed uniformly. Indeed, jittered sampling offers many advantages and can be computed efficiently using techniques described in [ABRA85]. Using a single z-value for each surface descriptor causes aliasing when primitives interpenetrate. This is a limitation for some datasets, but may not be sufficiently serious to justify the expense of handling it correctly (this is a detail that needs investigation).

Handling multiple surfaces per pixel and subpixel coverage masks significantly complicates rasterization and image-composition. The scan-line rasterizer must be able to determine rapidly whether a surface completely covers a pixel, and if not, to determine which subpixels the surface covers (this is particularly difficult at the corners of primitives, since two edges must be considered). The variable amount of information for each pixel, which can increase or decrease if a surface partially overlaps a pixel or if a surface completely obscures those behind it, complicates the design and analysis of the compositors. One must be able to guarantee that for the set of images likely to be encountered, serious bottlenecks do not arise in the composition tree. The FIFO queues at the inputs of each compositor are designed to help this problem. Also, the use of a binary tree of compositors, which means that pixel streams pass through only $\log_2$ *# of renderers* compositors), should help this problem significantly. This problem requires analysis very early on in the proposed work.

This antialiasing method also requires the presence of a pixel assembler. A single hardware unit is probably insufficient for this task, since it would have to resolve the visibility of every visible subpixel in the final image, an enormous computing requirement for complex images displayed at high frame rates. An alternative is to multiplex pixels between several pixel assemblers, so each assembler only processes a fraction of the pixels in the display.

Memory Requirements. Since rendering is performed in scan-line order, only a small amount of memory is required beyond the memory for storing the graphics database.

Scan-line systems typically transform all primitives into screen-space and sort them by scan-line (a process called *bucket-sorting*)—all before rasterizing. This can be a convenience for software algorithms, but is not a strict requirement. An alternative is to initially compute only the y-component of each transformed vertex. This suffices to determine which scan-line bucket each primitive falls into. The bucket can be used to store a pointer to the primitive in object-space, rather the entire transformed primitive, as is typically done. Storage for the sorted list is minimal—approximately 2 words per primitive.

The active primitive list also requires some additional storage, but only enough for the list of primitives active for a given scan line. For most databases this should be 10% of the primitives in the database or less.

Finally, no z-buffer is required, since rendering is performed in scan-line order. Only a small amount of buffering is required to even out temporary differences in processing rates between the renderer and the tree of compositors.

Latency. An increasingly important issue in interactive systems is image-generation latency. Latency is the time between when input values (such as joystick position) are sampled and when the final image is presented to the viewer. This issue is distinct from update rate, which is the time between successive images.

For interactive applications both frame rate and latency are important. Users are distracted by jerky motion from a slow update rate. Users are also bothered by slow response times produced by high latency. In certain applications, such as head-mounted displays or flight simulators, high latency can cause motion sickness [DEYO89].

Many high-performance graphics systems achieve speed through pipelining. For example, primitives of one frame can be transformed, while primitives from the preceding frame are rasterized, while primitives from the preceding frame are refreshed from a double-buffered frame buffer. Pipelining decreases the time between frames, but does not improve latency. The minimum latency possible occurs when the latency is exactly equal to the image-generation time.

A scan-line image composition system can approach this minimum. If y-sorting is performed using the approach described above (in the Memory Requirements section), only one scan line is transformed and rasterized at a time. Image-composition adds very little latency to the display process, since pixels flow through only a small number ($\log_2$ *# of renderers* compositors. The only latency beyond the image-generation time occurs because the frame-buffer can only switch banks during vertical retrace. Since video refresh is faster than frame rates (typically 60 – 72 Hz. vs. 20 – 30 Hz), this is still only a fraction of the image-generation time on average.

Volume Rendering. Volume rendering is becoming an increasingly important method for visualizing 3D data. Real-time volume rendering is even more daunting than real-time polygon rendering because of the much larger amount of data involved. Volume rendering calculations can be parallelized in a number of ways, some of which potentially can be accelerated using image-composition hardware. Here we briefly outline one method that maps well to the architecture of the prototype system:

The volume dataset is statically decomposed into a number of blocks (for simplicity, these may be cubes, but this is not a requirement for the method). Voxel elements within each block are assigned to graphics processor as shown in Figure 3.

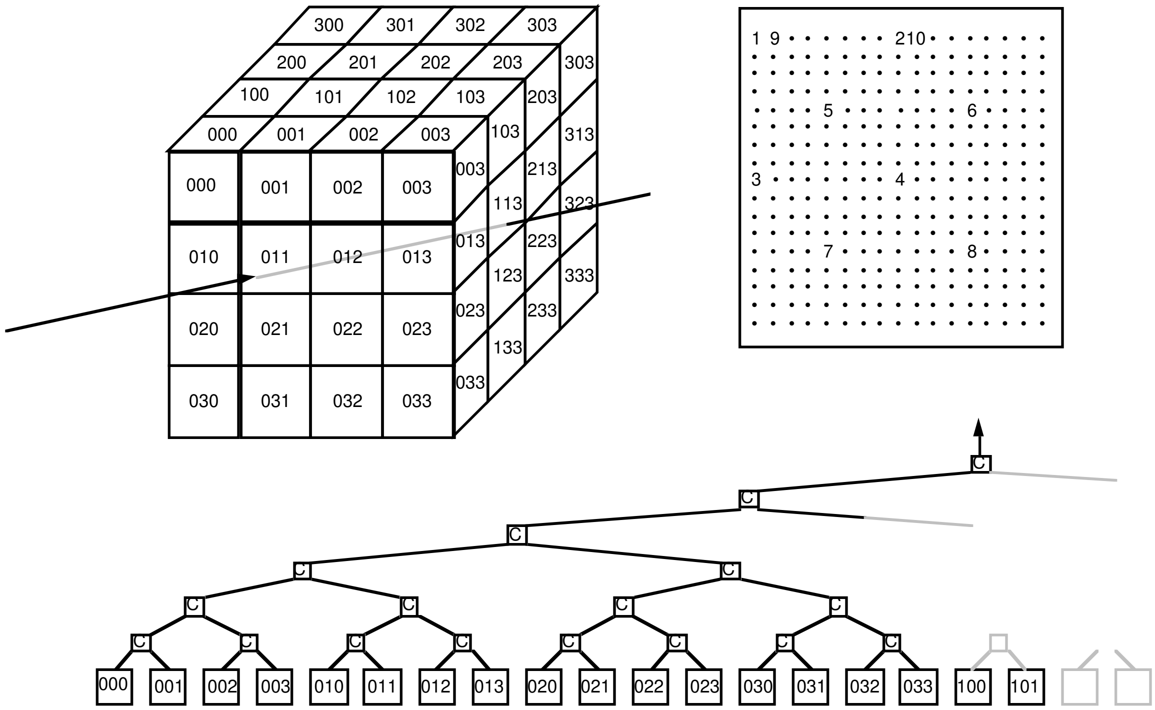

Figure 3: Decomposition of volume dataset into blocks and assignment to graphics processors.

To render each frame, the current viewing parameters are broadcast to each graphics processor. Rays are then cast into the volume array in a predetermined order (scan-line order is not good for load-balancing, as we will see later). Each graphics processor computes an aggregate color and opacity for its set of voxels, as seen by the current ray. It loads these, along with the z value where the ray intersects its block, into the image-composition tree. The composition tree then composites colors and opacities from adjacent blocks, until, at the root of the tree, a single color and opacity for the ray emerges.

The naive distribution of volume blocks to processors shown above has the useful property that at each composition node, color/opacity streams from each branch can be merged; there is no need to pass unmerged information further through the tree. This means that the data flowing through each tree branch is limited to just one color and opacity per ray.

One concern about this method of partitioning the dataset is that any given ray intersects only a fraction of the partition blocks. If the scene is rendered in scan-line order, some processors will be idle for long periods of time while others are extremely busy. A method of distributing the load is to send successive rays to different parts of the image. If the pattern in Figure 3 is used, a high probability exists that one of every eight rays will intersect each processors's block of voxels. If a queue is provided between the processor and the image-composition tree, the processor can quickly process rays which do not intersect its volume, and spend time on the few rays that do. This load-balancing scheme only works, of course, if the frame buffer can accept pixels out of scan-line order. An alternative rendering method proposed by Marc Levoy [LEVO90] is to have each renderer compute the "image" (color and opacity) of its voxels on each side its subvolume. These "textured" polygons can then be transformed and composited using the image-composition tree (to do this for a large range of viewpoints, the images must be calculated at different resolution levels, and a filtering method must be provided).

4. Research Plan and Schedule

I intend to demonstrate that the prototype system described above (or a modification to it) is practical and achieves some or all of the goals in Section 1. To do this, I propose the following research plan. Some of the steps have already been accomplished already. Others are dependent on the results of prelimary work and, therefore, must be somewhat flexible.

1. *Build rendering software testbed.* (prelimary version complete) This is important, since the family of rendering algorithms that will be efficient (and even possible) on the composite architecture will be constrained by hardware decisions. The rendering software testbed is flexible and easy to modify (rendering speed is a secondary concern here). The renderer subdivides the database into a variable number of partitions prior to rendering, and renders these partitions independently. It also allows different rendering algorithms and antialiasing methods to be tested and can sample pixels at variable resolution.

2. *Evaluate potential gains of variable-resolution sampling.* (prelimary results already) An open question is whether to treat pixels uniformly or to allow variable-resolution sampling, as described in the prototype system. The variable-resolution scheme counts on only a small percentage of the pixels of the screen requiring higher-resolution sampling (multiple visible surfaces per pixel). It offers the attractive possibility of generating high-quality images at nearly the same speed as unantialiased images, and provides a convenient way to incorporate transparent surfaces. If a large percentage of pixels requires higher resolution, however, the advantage this approach diminishes; a more parsimonious system may be built by simply supersampling in the traditional way. Simulations with the renderer testbed on a number of databases produce the following prelimary results:

Database	Room	Old Well	Space station
Number of polygons	249	356	3784
Screen size	128x128	256x256	512x512
Number of pixels	16384	65536	262144
Number of complex pixels	4025	13246	27105
Complex pixels/total pixels	0.25	0.202	0.103
Avg surfs per pixel	1.42	1.29	1.24
Avg surfs per complex pixel	2.72	2.44	3.33
Max surfs per complex pixel	10	12	15

Unfortunately, these results don't settle the question of which approach is better. More simulations with more complex datasets will cast further light on this. I have solicited datasets from a number of sources, and intend to gather statistics on them as soon as time permits. I expect that neither architectural approach will be a clear choice for all conditions.

3. *Develop criteria and algorithms for high-quality image generation.* Systems are only beginning to approach the threshold of real-time, high-quality image generation. To provide a basis for evaluating architectural alternatives, I propose to perform experiments to determine what levels of antialiasing, texture filtering, etc., are required to generate images of sufficient quality. I intend to render a number of representative images with varying degrees of sophistication in antialiasing: a) using a single point sample per pixel, b) supersampling on regular x,y grids of varying resolution (but with a single z value per pixel), c) supersampling with independent z values for each subsample, and d) stochastic super sampling. Since some aliasing is only visible when a series of consecutive frames is displayed, I will generate film loops of moving images to completely evaluate the image quality. I have ideas for an efficient form of successive refinement antialiasing using stochastic sampling, which I plan to develop. I also intend to look at the issues of prefiltering texture maps and look for algorithms to perform it efficiently (no ideas here yet).

4. *Attempt to map other rendering methods to the image-composition framework.* Different algorithms run more or less efficiently on different hardware architectures. An image-composition architecture is sufficiently different from current architectures and provides enough parallel processing capability that elegant implementations for many rendering techniques may exist (the volume rendering algorithm described above is one such example).

(The following steps are more tentative and depend on the results of previous steps)

5. *Simulate image-composition of component images.* If the steps above indicate that the variable resolution approach is potentially useful, I will write a program to simulate the image-composition tree described in Figure 2. In addition to combining pixel-streams correctly, this simulator will be parameterized to evaluate the flow-through characteristics of the image-composition tree. Parameters will include composition rates for pixels with different numbers of surface descriptors, the amount of FIFO buffering at each node, and the speed of the pixel assembler. I will estimate the bandwidth, contention, and utilization of system resources. The two simulators and the library of test objects will provide a testbed for developing implementations for the scan-line renderers, compositors, and pixel assembler.

6. *Investigate implementations for compositors.* Compositors may be simple or complicated, depending on whether variable resolution is used. In either case they must handle the bandwidth to generate images of desired complexity at a given update rate. There may be opportunities to make them programmable or general-purpose enough to allow volume rendering and other algorithms aside from displaying polygons.

7. *Investigate implementations for pixel assembler and/or final frame buffer.* If a variable resolution approach is used, a pixel assembler is needed. It must be able to assemble pixels at frame rates, but may include extra functionality to Phong-shade or texture pixels. These functions could also be included in a final accumulator frame buffer, which would probably be desirable even if the fixed-resolution approach is adopted. Substantial savings can be achieved by performing rendering tasks that are only performed once per pixel in the pixel assembler/accumulator frame buffer, rather than in individual renderers.

8. *Evaluate architectural alternatives, design, and (perhaps) implement a prototype system.* If the results of simulation are encouraging, I intend to begin a detailed design of the various parts of a prototype system (an entire system design would probably be infeasible except under the most optimistic circumstances). If simulation indicates that there are major difficulties with the prototype system as described, I intend to evaluate architectural alternatives. Many possibilities exist for hardware-assisted scan-line rendering and deferred evaluation of lighting/shading models.

I propose the following schedule for undertaking this research:

Date	**Activity**
3/89 – 9/89	Literature survey (performed in course of co-authoring Chapter 18 of [FOLE90]).
4/90 – 8/90	Build rendering software testbed and simulation platform for scan-line renderer and image-composition network. Gather statistics on sample images.
6/90	Proposal accepted
6/90 – 12/90	Characterize requirements for real-time, high-quality image-generation. Develop antialiasing and texturing algorithms. Investigate implementing other algorithms on an image-composition system.
9/90 – 12/90	Evaluate architectural alternatives based on results of above research and high-level simulation.
11/90 – 2/90	Converge on a desirable architecture, developing a high-level design for a prototype system.
9/90 – 1/91	Develop low-level design for interesting parts of architecture (perhaps implementing portions).
8/90 – 4/91	Prepare first draft of dissertation.
4/91 – 6/91	Clean up loose ends and prepare final draft of dissertation.
6/91 – 8/91	Thesis defense (some time in this range).

References

ABRA85 Abrams, G., L. Westover, and T. Whitted, "Efficient Alias-Free Rendering Using Bit-Masks and Look-Up Tables," *Computer Graphics* (Proceedings of SIGGRAPH '85), Vol. 19, No. 3, pp. 53–59.

AKEL88 Akeley, K. and T. Jermoluk, "High-Performance Polygon Rendering," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 239–246.

AKEL89 Akeley, K., "The Silicon Graphics 4D/240GTX Superworkstation," *IEEE Computer Graphics and Applications,* Vol. 9, No. 4, July 1989, pp. 71–83.

APGA88 Apgar, B., B. Bersack, and A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 255–262.

BEND89 Bender, C.F., *A Concurrent Visualization System,* proposal submitted to DARPA Information Science and Technology Office by the Ohio Supercomputer Center, 1224 Kinnear Rd., Columbus, OH 43212, May 1989.

BORD89 Borden, B.S., "Graphics Processing on a Graphics Supercomputer," IEEE Computer Graphics and Applications, Vol. 9, No. 4, July 1989, pp. 56–62.

BOUK70a Bouknight, W.J. and K.C. Kelly, "An Algorithm for Producing Half-tone Computer Graphics Presentations with Shadows and Movable Light Sources," *SJCC 1970,* AFIPS Press, Montvale, NJ, p. 1–10.

BOUK70b Bouknight, W.J., "A Procedure for Generation of Three-dimensional Halftoned Computer Graphics Representations," *Communications of the ACM,* Vol. 13, No. 9, September 1970, pp. 527–536.

BUNK89 Bunker, M. and R. Economy, "Evolution of GE CIG Systems," SCSD Document, General Electric Company, Daytona Beach, FL 32015, 1989.

CARP84 Carpenter, L., "The A-Buffer, an Antialiased Hidden Surface Method," *Computer Graphics* (Proceedings of SIGGRAPH '84), Vol. 18, No. 3, pp. 103–108.

CHUN89 Chung, J.C., M.R.Harris, F.P. Brooks, H. Fuchs, M.T. Kelley, J. Hughes, M. Ouh-young, C. Cheung, R.L. Holloway, M. Pique, "Exploring Virtual Worlds with Head-Mounted Displays," *Proc. SPIE Meeting on Non-Holographic True 3-Dimensional Display Technologies*, Vol. 1083, Los Angeles, Jan. 15-20, 1989.

CLAR80 Clark, J. and M. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *VLSI Design,* Q4, 1980, pp. 40–45.

COOK86 Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics,* Vol. 5, No. 1, January 1986, pp. 51–72.

CROW77 Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images," *Communications of the ACM,* Vol. 20, No. 11, November 1977, pp. 799–805.

CROW81 Crow, F.C., "A Comparison of Antialiasing Techniques," *IEEE Computer Graphics and Applications,* Vol. 1, No. 1, January 1981, pp. 40–48.

CROW84 Crow, F.C., "Summed-Area Tables for Texture Mapping," *Computer Graphics* (Proceedings of SIGGRAPH '84), Vol. 18, No. 3, pp. 207–212.

DEER88 Deering, M., S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 21–30.

DEME80 Demetrescu, S., "A VLSI-Based Real-Time Hidden-Surface Elimination Display System," Master's Thesis, Department of Computer Science, California Institute of Technology, 1980.

DEME85 Demetrescu, S., "High Speed Image Rasterization Using Scan Line Access Memories," *Proceedings of the 1985 Chapel Hill Conference on VLSI*, Rockville, MD, Computer Science Press, pp. 221–243.

DEYO89 Deyo, R. and D. Ingebretson, "Notes on Real-Time Vehicle Simulation," in *Implementing and Interacting with Real-Time Microworlds,* ACM SIGGRAPH '89 course notes.

DIPP85 Dippe, M.A.Z. and E.H. Wold, "Antialiasing Through Stochastic Sampling," *Computer Graphics* (Proceedings of SIGGRAPH '85), Vol. 19, No. 3, pp. 69–78.

DUFF85 Duff, T., "Compositing 3D Rendered Images," *Computer Graphics* (Proceedings of SIGGRAPH '85), Vol. 19, No. 3, July 1985, pp. 41–44.

EYLE88 Eyles, J., J. Austin, H. Fuchs, T. Greer, and J. Poulton, "Pixel-planes 4: A Summary," *Advances in Computer Graphics Hardware II* (1987 Eurographics Workshop on Graphics Hardware), Eurographics Seminars, 1988, pp. 183-208.

FORR85 Forrest, A.R., "Antialiasing in Practice," in *Fundamental Algorithms for Computer Graphics,* ed. R.A. Earnshaw. NATO ASI Series. Springer-Verlag, 1985, pp. 113–134.

FUCH77 Fuchs, H., "Distributing a Visible Surface Algorithm over Multiple Processors," *Proceedings of the ACM Annual Conference*, pp. 449–451.

FUCH79 Fuchs, H. and B. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics," *Proceedings of the 6th ACM-IEEE Symposium on Computer Architecture*, April, 1979, pp. 58–67.

FUCH81 Fuchs, H. and J. Poulton. 3rd Quarter, 1981. "Pixel-planes: A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI Design*, 2(3), pp. 20–28.

FUCH82 Fuchs, H., S.M. Pizer, E.R. Heinz, S.H. Bloomberg, L. Tsai, and D.C. Strickland, "Design of and Image Editing with a Space-Filling Three-Dimensional Display Based on a Standard Raster Graphics System," *Proceedings of SPIE,* Vol. 367, August 1982, pp. 117–127.

FUCH85 Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-planes," *Computer Graphics* (Proceedings of Siggraph '85), Vol. 19, No. 3, pp. 111–120.

FUCH89 Fuchs, H., J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics* (Proceedings of SIGGRAPH '89), Vol. 23, No. 3, pp. 79–88.

FUSS82 Fussel, D. and B. D. Rathi, "A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons." *Graphics Interface '82*, 1982, pp. 373–380.

GHAR88 Gharachorloo, N., S. Gupta, E. Hokenek, P. Balasubramanian, B. Bogholtz, C. Mathieu, and C. Zoulas, "Subnanosecond Pixel Rendering with Million Transistor Chips," *Computer Graphics* (Proceedings of Siggraph '88), Vol. 22, No. 4, pp. 41–49.

GHAR89 Gharachorloo, N. and S. Gupta, "A Characterization of Ten Rasterization Techniques," *Computer Graphics* (Proceedings of SIGGRAPH '89), Vol. 23, No. 3, July 1989, pp. 355–368.

GLDW88 Goldwasser, S.M., R.A. Reynolds, D.A. Talton, and E.S. Walsh, "Techniques for the Rapid Display and Manipulation of 3-D Biomedical Data," *Comp. Med. Imag. and Graphics*, Vol. 12, No. 1, 1988, pp. 1–24.

GRIM89 Grimes, J., L. Kohn, and R. Bharadhwaj, "The Intel i860 64-Bit Processor: A General-Purpose CPU with 3D Graphics Capabilities," *IEEE Computer Graphics and Applications,* Vol. 9, No. 4, July 1989, pp. 85–94.

KEDE84 Kedem, G. and J. L. Ellis, "The Raycasting Machine," *Proceedings of the 1984 International Conference on Computer Design*, 1984, pp. 533–538.

LEVO89 Levoy, M., "Design for a Real-Time High-Quality Volume Rendering Workstation," *Proceedings of Volume Visualization Workshop,* Dept. of Computer Science, University of North Carolina at Chapel Hill, May 18-19, 1989, pp. 85–90.

LEVO90 Levoy, M., *Personal communication.*

MCMI87 McMillan, L., "Graphics at 820 MFLOPS," *ESD,* Vol. 17, No. 9, September 1987, pp. 87–95.

MOLN88 Molnar, S., "Combining Z-buffer Engines for Higher-Speed Rendering," *Advances in Computer Graphics Hardware III*, Eurographics Seminars, 1988.

NAKA89 Nakamae, E., T. Ishizaki, T. Nishita, and S. Takita, "Compositing 3D Images with Antialiasing and Various Shading Effects," *IEEE Computer Graphics and Applications,* Vol. ?, No. ?, March 1989, pp. 21–29.

PORT84 Porter, T. and T. Duff, "Compositing Digital Images," *Computer Graphics* (Proceedings of SIGGRAPH '84), Vol. 18, No. 3, July 1984, pp. 253–259.

ROMA89 Romanova, C. and U. Wagner, "A VLSI Architecture for Anti-Aliasing", *Advances in Computer Graphics Hardware IV*, Eurographics Seminars, 1989.

ROMN69 Romney, G.W., G.S. Watkins, and D.C. Evans, "Real Time Display of Computer Generated Half-Tone Perspective Pictures," *Proceedings 1968 IFIP Congress,* North Holland Publishing Co., 1969, p. 973.

SCHA83 Schacter, B., *Computer Image Generation*, John Wiley & Sons, Inc., New York, 1983.

SCHU80 Schumacker, Robert, "A New Visual System Architecture," *Proceedings of the Second Interservice/Industry Training Equipment Conference,* Salt Lake City, Utah, 16-20 November 1980.

SCHN88 Schneider, B.O., "PROOF: An Architecture for Rendering in Object-Space", *Advances in Computer Graphics Hardware III*, Eurographics Seminars, 1988.

SHAW88 Shaw, C.D., M. Green, and J. Schaeffer, "A VLSI Architecture for Image Composition," *Advances in Computer Graphics Hardware III*, Eurographics Seminars, 1988.

SGI90 Silicon Graphics Computer Systems, Vision Graphics System Architecture, Mountain View, CA 94039-7311, February 1990.

SUTH65 Sutherland, I.E., "The ultimate display," *Proceedings of the 1965 IFIP Congress,* Vol. 2, 1965, pp. 506–508.

SUTH68 Sutherland, I.E., "A Head-mounted Three Dimensional Display," *FJCC 1968,* Thompson Books, Washington, D.C., pp. 757–764.

SUTH74 Sutherland, I.E., R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden-Surfae Algorithms," *Computing Surveys,* Vol. 6, No. 1, March 1974, pp. 1–55.

WATK70 Watkins, G., "A Real-Time Visible Surface Algorithm," University of Utah Computer Science Department, UTEC-CSc-70-101, June 1970, NTIS AD-762

004.

WEIN81 Weinberg, R., "Parallel Processing Image Synthesis and Anti-Aliasing," *Computer Graphics* (Proceedings of SIGGRAPH '81), Vol. 15, No. 3, pp. 55–61.

WHIT82 Whitted, T. and D.M. Weimer, "A Software Testbed for the Development of 3-D Raster Graphics Systems," *ACM Transactions on Graphics,* Vol. 1, No. 1, January 1982, pp. 44–58.

WILL83 Williams, L., "Pyramidal Parametrics," *Computer Graphics* (Proceedings of SIGGRAPH '83), Vol. 17, No. 3, pp. 1–11.

WYLI67 Wylie, C., G.W. Romney, D.C. Evans, and A.C. Erdahl, "Halftone Perspective Drawings by Computer," *Fall Joint Computer Conference 1967*, Thompson Books, Washington D.C., p. 49.