Reprinted from: *Advances in Computer Graphics Hardware III* (Proceedings of the 1988 Eurographics Workshop on Graphics Hardware), Eurographics Seminars, 1988, pp. 171-182.

Combining Z-buffer Engines for Higher-Speed Rendering

Steven Molnar

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

Abstract

Described is a hardware architecture for combining the outputs of a number of z-buffer rendering engines to achieve higher performance than is possible with a single renderer. It allows a combination of renderers to achieve the same price/performance ratio as the individual renderers that compose it, and can be extended to create systems with arbitrarily high performance.

The described architecture is based on a fusion of scan-line rendering and the conventional z-buffer algorithm. The frame buffers of several z-buffer engines are modified to scan out z-values as well as color values. Multiplexing devices combine the z/color streams from each pair of frame-buffers. These z/color streams are then combined by further multiplexers, creating a binary tree that funnels the z/color information from the many conventional frame buffers into a single z/color stream. The color stream is then used to drive a standard display device.

The proposed architecture allows rendering rates of millions and even tens of millions of polygons per second. The basic architecture can be extended with additional hardware to perform antialiasing and texture-mapping.

1. Introduction

The performance of raster graphics hardware has increased dramatically over the past several years. Machines now exist that render and shade hundreds of thousands of polygons per second. Graphics engines are relying more and more heavily on parallelism to speed the rendering process. As levels of parallelism increase in future years, one can expect architectures to come closer to physical limitations such as the speed of light and maximum memory bandwidths. When this point grows near, performance gains will become more and more difficult.

This paper explores a way to sidestep the above problem by combining the outputs of multiple z-buffer rendering engines to achieve higher performance than is possible in a single renderer. The resulting composite system will have approximately the same performance/price ratio as its component renderers, and can be used to build systems with arbitrarily high performance.

.

Figure 1 shows an overview of the proposed system. In this example eight rendering engines, each capable of rendering 400,000 Gouraud-shaded triangles per second, are harnessed together, providing a net rendering speed of 3.2 million triangles per second. Because of the way that images are combined, no performance degradation occurs no matter how many frame buffers are used to achieve the performance.

This paper describes the method used to combine outputs from multiple frame buffers, analyzes the cost effectiveness and limitations of these composite systems, and proposes extensions to allow more sophisticated rendering techniques such as antialiasing and texture-mapping.

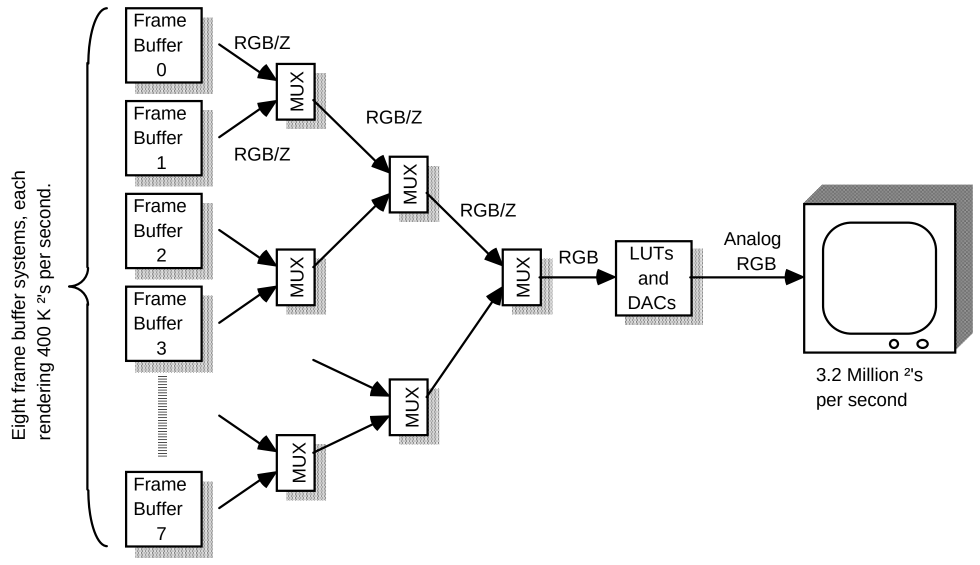


Figure 1. Composite system for displaying 3.2 million triangles per second

The work in this paper follows related work by a number of researchers:

[Demetresc80] and [Deering88] describe pipelined architectures for rendering shaded polygons where one polygon is assigned per processor. Each processor contains input and output RGB and z ports. The image is traversed pixel-by-pixel in raster scan order. Each processor compares the z value at its input with the z value computed for its polygon. If the polygon covers the pixel and the computed z value is smaller than the input, the processor transmits the RGB and z values it just calculated. Otherwise it simply transmits the RGB/z at its input. Demetrescu designed chips based on his ideas, but never integrated them into a complete system. Deering et. al. describe an overall system architecture, including another custom processor to implement sophisticated lighting models, but did not build a prototype system.

[Fussel82] also proposes an architecture in which one polygon is assigned per processor. Rather than configuring the processors in a pipeline, as in the above architecture, Fussel introduces the idea of combining images using a binary tree of comparators.

[Kedem84] proposes a related architecture for ray-casting Constructive Solid Geometry (CSG) objects. In this system, primitives of a CSG tree are assigned to custom processors.

Other custom processors at the interior nodes of the CSG tree perform z-comparisons and in/out classifications required for CSG. Kedem and Ellis recently built a prototype of their system.

[Leray87] proposes a method for combining two z-buffered images to form a composite image in a manner analogous to chroma-keying. He proposes an unpipelined version of the the same color/z comparison hardware we describe in the following section. His paper does not extend the idea to a system composed of numerous frame buffers.

2. Scan-line Composition of Z-buffer Images

Scan-line composition of z-buffer images is an extension of the basic z-buffer algorithm, in which z-values are used to determine which components of an image are visible [Foley82, pp. 560–561]. If a database is distributed across multiple rendering engines, with each engine using the same modeling and viewing transformations, a composite scene can be generated by comparing corresponding pixels in each frame buffer and choosing the pixel with the smallest z-value for display in the composite image.

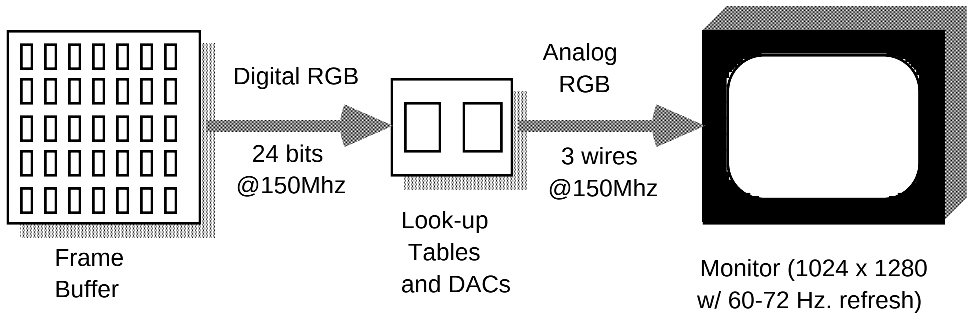


Figure 2. Conventional high-resolution video system

The video scan-out mechanism of a conventional frame buffer provides a ready-made method to do this. Normally RGB values for each pixel are scanned out in raster order and sent to the display device (Figure 2). If we modify the frame buffer to scan out z values as well as RGB values, we can combine the RGB values from two frame buffers with a simple piece of hardware. This hardware part is shown schematically in Figure 3. It compares incoming z-values from two z-streams and passes the z and RGB values of the visible pixel. It can easily be built from off-the-shelf components for around US $100 (plus board and connector costs).

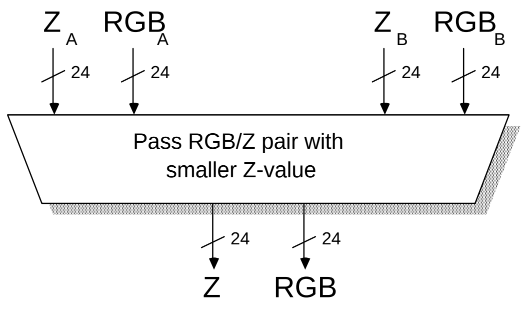


Figure 3. Z-comparator / multiplexing device

The composite machine generates images as follows: first, each rendering engine transforms its portion of the database into screen space and renders it into its z/color buffer. Next, the frame-buffers synchronously scan out z and color values into the tree of multiplexing devices, where they are successively combined until a single RGB/z stream emerges at the root. Finally, this RGB/z stream is used to produce the composite image by feeding its RGB portion into color look-up-tables and video DACs. Figure 1, on the previous page, shows the entire display process.

The following sections discuss aspects of the architecture and display process in more detail.

2.1. Database Distribution

In a composite system each renderer is responsible for rendering a fraction of the database. For good renderer utilization, the database should be distributed evenly across the renderers. If a database is hierarchical one must decide whether to distribute primitives *horizontally* or *vertically*.

Horizontal distribution means replicating the entire hierarchical structure at each renderer and scattering the primitives at each node across all of the renderers. This scheme has the advantage of making load balancing among renderers automatic (assuming that primitives can be rendered at a uniform rate and that each node contains a large number of primitives), but has the disadvantage of requiring extra space and time to store and traverse multiple copies of the database hierarchy. For databases with deep hierarchies, these penalties can be especially costly.

Vertical distribution assigns entire subtrees to renderers. A renderer, therefore, stores only the portions of the hierarchy for which it is responsible. This scheme saves space and reduces the time spent traversing the hierarchy, but makes load balancing among renderers more difficult, since subtrees in the hierarchy are likely to contain different numbers of primitives, and changes in viewing parameters can cause whole subtrees to lie entirely outside of the viewing frustum. To overcome these limitations, a dynamic load calculation must be performed on the database, and heuristics used to allocate subtrees to renderers. The overhead of allocating the database in this manner could easily overwhelm any savings in database traversal time.

2.2. Scanning Out RGB and Z-values

The frame buffers in most commercial rendering systems are built out of dynamic memory parts. Single-ported DRAMs are generally used to store z-values and other pixel data not for display, while dual-ported VRAMs are used to store the RGB values. An easy way scan out z-values is to store them in VRAMs in the same manner as RGB values. VRAM memory densities approach DRAM densities, and the same sequencing logic that drives the RGB scanout circuitry can drive the z scanout circuitry. A method for synchronizing the scanout circuitry on each of the boards is needed. A straightforward way to do this would be to provide a global, synchronous scanout clock to each frame buffer board. Note that if rendering is to occur concurrently with video display, the z-buffer must be double-buffered in addition to the color buffer. This requires extra memory. These changes will require modifications to the frame-buffer board, but the changes will largely be confined to the video output portion.

2.3. The Comparator/Multiplexing Device

The comparator/multiplexing device merges two RGB/z data streams into one. Even though simple, the multiplexing device must run at very high speeds (Å 150 MHz for a 1024x1280 frame monitor refreshing at 60 Hz). The multiplexer can be pipelined by sending z-values one clock cycle ahead of RGB values. It can be implemented using off-the-shelf ECL components or a custom gate array. Using Fairchild F100K ECL parts a comparator/multiplexer such as the one diagrammed in Figure 4 can be built for around US $110[1] (plus board and connector costs).

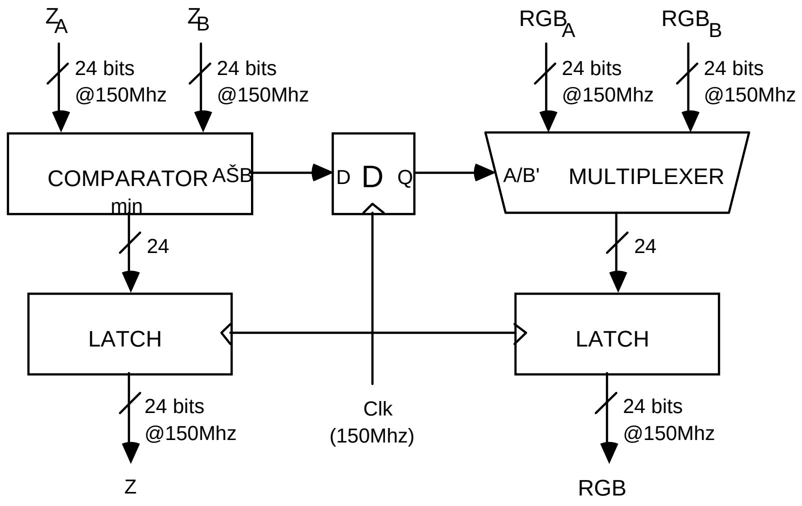


Figure 4. Pipelined Z-comparator / multiplexing device

3. The quest for higher performance

Computer graphics rendering hardware has enjoyed the same exponential-increase in performance as general-purpose computers. Commercial raster graphics systems began appearing in the late 1970's. The earliest of these were general purpose processors coupled with frame buffer memories, and had very low performance compared with today's standards. Hewlett-Packard and Silicon Graphics in 1986 each delivered machines capable of rendering 15-25 thousand triangles per second. This year Ardent Computer Corp., Silicon Graphics Computer Systems, and Stellar Computer Inc. each are delivering graphics supercomputers boasting performances of 100-200 thousand polygons per second [Manuel88]. Shaded graphics rendering speeds have been increasing by nearly a factor of two every year.

But can the trend continue? Clock speeds of the latest generation graphics engines lie in the 10-50 Mhz range. Certainly some increase is possible as technologies mature, but wiring delays become more and more significant as clock speeds increase. The speed of light places a fundamental limit on how fast signals can propagate in any computer or graphics system, and technology is rapidly approaching that limit. We can expect only a modest increase in performance due to increased clock speeds.

[1]Price of 6 Fairchild F100166 and 12 Fairchild F100155 ECL parts (Hamilton-Avnet Electronics, Raleigh, NC, prices quoted 30/8/88).

3.1 Parallelism

What about parallelism? The latest generation graphics engines already contain many levels of parallelism. Consider, for example, the Silicon Graphics Iris 4D/70GT and the Stellar GS1000 graphics workstations, both released this year. The Iris is capable of rendering 100,000 Gouraud-shaded, z-buffered triangles per second. It uses a pipeline of 5 geometry engines for its "front-end", and a total of 32 processors for scan-conversion and video scanout. The Stellar GS1000 renders 150,000 Gouraud-shaded triangles per second. It has four vector floating point engines for front-end calculations and 16 processors arranged in a "footprint" pattern for scan-conversion and pixel operations.

The fact that these graphics engines employ many layers of parallelism is no coincidence. Rendering 100 thousand triangles per second requires 20 MegaFlops of compute power and a memory bandwidth of over 30 Megawords per second–very demanding for a uniprocessor. Higher-speed rendering requires proportionately more performance. Clearly parallelism is necessary, but how can it best be applied to get near-linear performance increases from added processors?

Existing machines apply parallelism in two locations: the *front-end*, where the display list is traversed and primitives are transformed from object space to screen space, and the *back-end*, where primitives are scan-converted and rendered into frame-buffer memory.

Front-end parallelism appears to be a fundamental requirement for high-performance rendering architectures. [Torborg87] discusses some of the problems and issues involved in front-end architectures; further discussion is beyond the scope of this paper.

Back-end parallelism has proven effective in many systems, such as the Stellar GS1000 [Apgar88], the Iris 4D/70GT [Akeley88], the Ardent Titan, and Pixel-planes [Eyles87], but suffers from decreasing returns as it is extended further and further. In all of these schemes clock speeds and bandwidth into frame buffer memory place strangleholds on maximum possible performance, regardless of the number of processors.

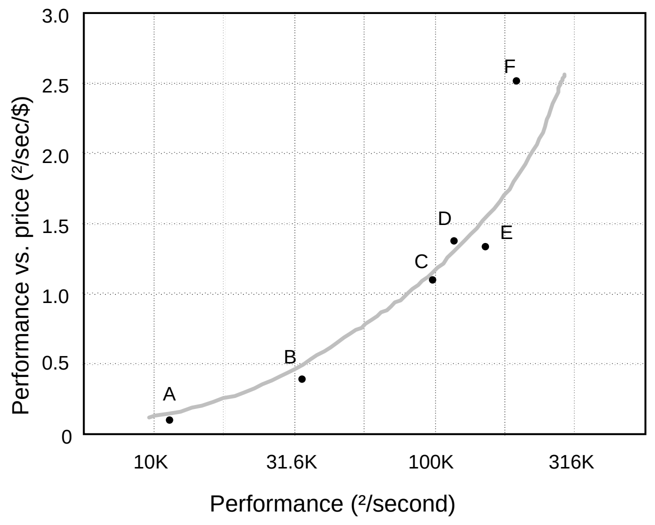
The architecture proposed in this paper applies parallelism at a later stage in the display pipeline–after the image is generated in the individual renderers. Since a constant amount of information is accumulated for each pixel of each frame buffer (one z and color value), no matter how many primitives contributed to the pixel, pixel information from several renderers can be combined without regard to scene complexity or uniformity. This only requires a single z-compare and setting of a multiplexer at each level in the multiplexing tree, so it can be done at video scanout speeds.

Because the tree depth is small ($\log_2$ # of renderers) as is the pipeline delay for each stage in the tree (2*the video clock period), this scheme adds no appreciable latency to the frame display time, a major problem for systems that use pipelining to achieve high update-rates. These properties enable one to build systems with arbitrarily high performance.

3.2 Economics

The cost of the composite system described above is equal to the sum of the costs of the individual renderers (with enhancements made for scanning out z-values) plus the costs of the multiplexing devices. A composite rendering system with 2^n frame buffers requires 2^n -1 multiplexing devices. Multiplexing devices are very cheap: ECL parts cost $110 per device; board area and wiring might add an additional $100 per device. The smallest rendering/z-buffer systems cost several thousand dollars. Therefore, the price of multiplexing devices is a negligible component of the price of a composite system, and the overall cost-effectiveness of a composite system will be determined by the cost-effectiveness of the individual renderers.

Figure 5 shows performance/price ratios for a number of currently-available rendering systems. One can see from the graph that the highest-performance renderers today have a higher performance/price ratio than smaller systems. This means that current architectures are not yet pushing physical limitations and that one cannot hope to build a composite system competitive with existing rendering systems. However, if one demands more performance than is possible in a single system, or one waits several years until rendering architectures approach closer to theoretical limits and the performance/price curve begins to turn downward, composite architectures become economically feasible.



Machine	Performance	Price (US $)
A	12,000	64,000
B	38,000	91,000
C	100,000	84,900
D	136,000	96,000
E	150,000	113,500
F	200,000	79,000

Machines in alphabetical order are: Alliant/Rastertech GX4000, Ardent Titan, HP 9000 Model 835 TurboSRX, Silicon Graphics 4D/70GT, Stellar GS1000, and the Sun TAAC

Figure 5. Performance/price ratios for representative systems[2]

4. Advanced Rendering Algorithms

The limitations imposed by the composite architecture are two-fold:

- Only a (small) constant amount of information from each pixel can be scanned out and used for final image generation.
- Hidden-surface elimination must be done by z-buffer alone.

[2]Machines are not associated with letters because the purpose of this graph is to show trends in performance vs. price, not to compare machines. For a fair comparison, standard benchmarks and configurations are needed.

These requirements preclude rendering scenes with shadows, true transparency, or scenes defined by set operations (as in the display of Constructive Solid Geometry objects). Nevertheless, several important rendering algorithms can be implemented by placing a frame buffer with pixel-processing capabilities between the root of the composition tree and the video look-up-tables. These algorithms include antialiasing, texture-mapping, and a stochastic form of transparency. This scheme generalizes into a method of factoring pixel-oriented computations out of the individual renderers and onto this frame buffer processor to increase rendering throughput.

4.1 Antialiasing

Aliasing results from two steps in the rendering process: scan conversion and depth-resolution. Because scan conversion is performed at the individual renderers, the renderers can remove scan-conversion aliasing artifacts. Depth-resolution, however, is divided between the renderers and the z-composition tree, so individual renderers will not be able to remove all of the z-buffer artifacts. Consequently, a more general antialiasing technique is needed.

Two basic approaches toward antialiasing exist: explicit calculation of pixel-coverage (A-buffer techniques), and supersampling techniques.

The A-buffer algorithm [Carpenter84] produces high-quality antialiased images, but requires an arbitrarily long list of partially obscuring surfaces to be maintained for each pixel. This is obviously not suitable for a composite architecture in which only a fixed, small amount of data can be stored at each pixel.

Simplifications and variants of the A-buffer scheme are possible (see [Duff85]). One appealing simplification, which would incur little speed penalty, is to truncate the list of partially-visible surfaces at two. This would obviate the need for the partial coverage mask and its associated computations. Two data fields would be added to the RGB/Z fields for a pixel: *alpha* (a one-byte value included with the pixel's RGB value), and a second RGB value. Alpha specifies the fraction of the pixel covered by the closest surface, and the second RGB value specifies the next-closest surface's color. After a frame is rendered, pixels have their color values blended with the color values of the next-closest surface according to the following equation:

$$RGB = \alpha * RGB_{\text{closest surface}} + (1 - \alpha) * RGB_{\text{next closest surface}}$$

This technique will produce acceptable results for most pixels, but will fail when the second surface does not completely cover the pixel. Another difficulty arises in relation to the composite architecture: when pixel streams merge, there is no way to determine which surface should be the next-closest surface: the next-closest surface from the same pixel stream, or the closest surface from the other pixel stream. Of course, extra hardware could be added to sort next-closest surfaces, but a proliferation of parts and wires would result and "bad" pixels would still be produced.[3]

[3][Shaw88] (published concurrently with this paper) describes a VLSI implementation of the comparator/multiplexor that addresses the problem of z-buffer aliasing. These *compositors* perform antialiasing using a simplified version of Duff's image composition algorithm [Duff85]. Shaw's architecture allows antialiasing to be performed without the prohibitive speed penalty of supersampling, but requires more extensive modifications to the z-buffer renderers and occasionally produces miscolored pixels.

Supersampling produces images of uniformly high quality, but at great computational expense; an image with N samples per pixel takes N times as long to render as a simple image. [Fuchs85] and [Eyles87] present a way to supersample an image in incremental fashion, presenting the raw image first at full speed, then refining it by performing a weighted average with further samples. This is done for as long as the image remains stationary. This technique can be implemented on a composite system by adding an accumulator frame buffer after the pixel streams are combined. This frame buffer must have the ability to calculate linear combinations of color values at frame rates. This approach seems well-suited for interactive applications, allowing renderers to achieve maximum update rates while the scene is changing and to antialias while the scene is stationary.

4.2 Texture Mapping

Texture mapping can be performed in either of two locations: at individual renderers, or after the pixel streams have been combined. Each location has advantages and disadvantages.

Texture mapping at the renderers requires no additional hardware, but requires texture maps to be stored multiple times. This is expensive for large or numerous texture maps. It also wastes processing resources, since texture calculations are performed on pixels that do not contribute to the final image.

When texturing within the pixel stream, the texture space coordinates (u and v) and a texture tag replace the RGB value at a pixel. A special hardware processor performs texture look-ups for pixels needing texturing. Note that a simple table lookup will not suffice here, since aliasing artifacts are very pronounced with textures. To combat these, a technique such as summed area tables [Crow84] or mip maps [Williams83] must be used. A disadvantage of this technique is that it would be very difficult to build a texturing engine that could operate at video speeds. Pixel scanout would likely have to proceed at some slower speed determined by the texturing hardware.

4.3 Transparency

Transparency is difficult to handle in a z-buffer system since rendering transparent objects requires primitives to be sorted and rendered in order. One method to approximate transparency is to use a random screen to disable pixels on a transparent object. This technique has been implemented on Pixel-planes 4 [Eyles87]. It can be annoying because it adds noise to the image, but the noise becomes less objectionable after a number of uncorrelated images have been averaged together, which occurs in the above antialiasing scheme.

4.4 End of Frame Calculations

The above algorithms require an extra piece of hardware at the root of the z-composition tree. One can conceive of a system in which this hardware unit is made more general purpose, allowing other end-of-frame calculations to be performed there, such as lighting, shading, and environment mapping calculations. Such a system would factor out a significant portion of the rendering task from the individual renderers, increasing the effective speed of the renderers.

A prototype for such a general purpose frame buffer/ALU exists: the UNC Pixel-planes system [Eyles87]. Pixel-planes contains a 512x512 frame buffer built of custom VLSI chips. Each pixel in the frame buffer contains 72 bits of local memory and a one-bit processor. To serve as an end-of-frame processor, the Pixel-planes chips would have to be modified to allow pixel values to be scanned in, as well as out, but some features of the current chips could be eliminated. Such a system could achieve very efficient utilization of processing resources, in addition to rendering at arbitrarily high speeds.

5. Conclusion

We have shown how a number of z-buffer engines may be combined to form a composite rendering system with arbitrarily high performance and no added frame latency. The composite architecture requires only minor modifications to the z-buffer systems and the addition of a tree of comparator/multiplexing devices to combine the RGB/Z outputs of the multiple renderers.

Since the comparator/multiplexing devices are very inexpensive, a composite system will have approximately the same performance/price ratio as the renderers that compose it. This makes it easy to determine when a composite system is cost-effective. At the present time, performance/price ratios increase with hardware performance, so composite systems only make sense if they render faster than any current system. In the future, as renderer technology pushes closer to fundamental limits, the performance/price curve of rendering systems will certainly turn downward, making the composite architecture competitive with systems of the same performance.

Advanced rendering techniques such as antialiasing, texturing, and transparency can be performed by adding an extra frame buffer/pixel processor at the root of the z-composition tree. This provides the potential for factoring end-of-frame computations out of the individual renderers.

Acknowledgements

I wish to thank the entire Pixel-planes team for their ideas and many useful suggestions, particularly Henry Fuchs, Greg Turk, and John Eyles, who reviewed the paper and provided many helpful insights. This work was supported in part by the Office of Naval Research, contract no. N00014-86-K-0680, NSF grant no. MIP-8601552, and U.S. DARPA/ISTO order no. 6090.

References

[Akeley88] Akeley, K. and T. Jermoluk, "High-Performance Polygon Rendering," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 239–246.

[Apgar88] 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.

[Carpenter84] Carpenter, L., "The A-buffer, an Antialiased HIdden Surface Method," *Computer Graphics*

(Proceedings of SIGGRAPH '84), Vol. 18, No. 3, pp. 103–108.

[Crow84] Crow, F., "Summed-Area Tables for Texture Mapping," *Computer Graphics* (Proceedings of SIGGRAPH '84), Vol. 18, No. 3, pp. 207–212.

[Deering88] 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.

[Demetres80]Demetrescu, S., "A VLSI-Based Real-Time Hidden-Surface Elimination Display System," Master's Thesis, Department of Computer Science, California Institute of Technology, 1980.

[Duff85] Duff, T., "Compositing 3-D Rendered Images," *Computer Graphics* (Proceedings of SIGGRAPH '85), Vol. 19, No. 3, pp. 41–44.

[Eyles87] Eyles, J., J. Austin, H. Fuchs, T. Greer, and J. Poulton, "Pixel-planes 4: A Summary," *Advances in Graphics Hardware 2: Proceedings of the Eurographics '87 Second Workshop on Graphics Hardware*.

[Foley82] Foley, J., and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1982.

[Fuchs85] Fuchs, H., J. Goldfeather, J.P. Hultquist, S. Spach, J.D. Austin, F.P. Brooks, J.G. 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.

[Fussel82] 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.

[Kedem84] Kedem, G. and J. L. Ellis, "Computer Structures for Curve-Solid Classification in Geometric Modeling," 1984. Technical Report TR84-37, Microelectronic Center of North Carolina, Research Triangle Park, NC.

[Leray87] Leray, P., "3D Z-Key: An Enhanced Version of Chroma-Key," *Advances in Graphics Hardware 2: Proceedings of the Eurographics '87 Second Workshop on Graphics Hardware*.

[Manuel88] Manuel, T., "Graphics Visualization is a Real Eye Opener," *Electronics*, Vol. 61, No. 6, March 17, 1988, pp. 91–92.

[Shaw88] Shaw, C., M. Green, and J. Schaeffer, "A VLSI Architecture for Image Composition," *Advances in Graphics Hardware 3: Proceedings of the Eurographics '88 Third Workshop on Graphics Hardware*.

[Torborg87] Torborg, J. "A Parallel Processor Architecture for Graphics Arithmetic Operations." *Computer Graphics* (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, pp. 197–204.

[Williams83] Williams, L., "Pyramidal Parametrics," *Computer Graphics* (Proceedings of SIGGRAPH '83), Vol. 17, No. 3, pp. 1–11.