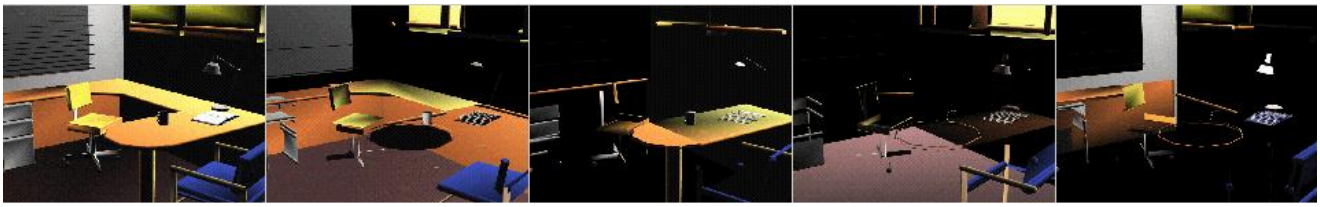
Scalable Distributed Visualization Using Off-the-Shelf Components

Alan Heirich*
Tandem Laboratories
Compaq Computer Corporation

Laurent Moll†
Systems Research Center
Compaq Computer Corporation



Abstract

This paper describes a visualization architecture for scalable computer systems. The architecture is currently being prototyped for use in Beowulf-class clustered systems. A set of OpenGL frame buffers are driven in parallel by a set of CPUs. The visualization architecture merges the contents of these frame buffers by user-programmable associative and commutative combining operations. The system hardware is built from off-the-shelf components including OpenGL accelerators, Field Programmable Gate Arrays (FPGAs), and gigabit network interfaces and switches. A second-generation prototype supports 60 Hz operation at 1024 × 1024 pixel resolution with interactive latency up to 1000 nodes.

CR Categories: B.7.1 [Integrated circuits]: Types and design styles—Gate arrays; C.2.5 [Computer-communication networks]: Local and wide-area networks—High-speed; D.1.3 [Programming techniques]: Concurrent programming—Parallel programming; I.3.1 [Computer graphics]: Hardware architecture—Parallel processing; I.3.2 [Computer graphics]: Graphics systems—Distributed/network graphics

Keywords: FPGA, OpenGL, visualization, cluster, Beowulf, gigabit, fat-tree

1 Introduction

Interactive visualization is invaluable to many users of large scale computer systems. The human nervous system absorbs information quickly when it is presented in the form of colored, shaded, and moving imagery. Such imagery requires high frame rates, and this requirement is difficult to meet for large data sets, or data sets that are distributed across multiple computers. Unfortunately these characteristics are typical of "grand challenge" problems and many other important computations. Regardless of whether the goal is to visualize a simulation as it is computed, or to analyze data in a post-processing step, the amount of data can overwhelm the largest conventional rendering systems [13, 17].

One approach to supporting these challenging requirements is to combine the capabilities of a set of relatively modest rendering systems. Our visualization architecture takes this approach and works by combining the outputs of many rendering systems running in parallel. This approach is scalable because the requirements for computation and data movement are determined by image size rather than by data size [1, 4, 8, 12, 16].

2 Image Combining: Theory

Call $\mathcal{S} = \{A, B, \ldots Z\}$ a set of images, each image an $n-$tuple of pixels $A = (a_1, a_2, \ldots, a_n)$, and each pixel a tuple of components $a_i = (r_{a_i}, g_{a_i}, b_{a_i}, z_{a_i}, \alpha_{a_i}, stencil_{a_i})$. The goal of our visualization architecture is to compute a pixel-by-pixel merge

$$A \oplus B \oplus C \oplus \cdots \tag{1}$$

for various binary operators $\oplus$. For example, if $D = A \oplus B$ then $D = \{(d_1, \ldots, d_n) : d_i = a_i \oplus b_i\}$. We'll call these binary operators *combining operators* and their functional descriptions *combining functions*. An example of a combining function is the *Z-comparison* function,

$$\begin{aligned} Z(a,b) &\equiv Z((r_a, g_a, b_a, z_a), (r_b, g_b, b_b, z_b)) \\ &= (r_a, g_a, b_a, z_a) \quad \text{if } z_a < z_b \\ &= (r_b, g_b, b_b, z_b) \quad \text{if } z_a > z_b \\ &= \text{undefined} \quad \text{if } z_a = z_b. \end{aligned} \tag{2}$$

In OpenGL the undefined condition may be defined in various ways to yield a semi-commutative combining function[1]. Responsibility for resolving the semi-commutativity is left to the application programmer. Commutativity implies that operands may be interchanged so that $Z(a,b) = Z(b,a)$. This property persists under function composition so that

$$Z(a, Z(b,c)) = Z(Z(c,b), a). \tag{3}$$

*19333 Vallco Parkway, Cupertino, CA 95014
†130 Lytton Ave, Palo Alto, CA 94301

[1] A function is *semi-commutative* if it is commutative on all but a finite subset of its domain.

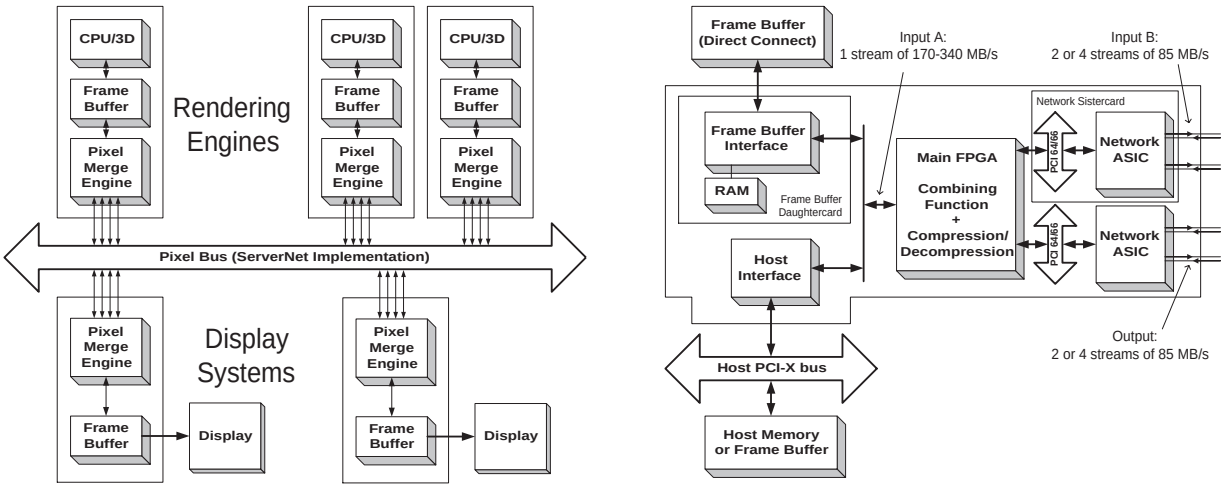


Figure 1: System architecture (left) and Pixel Merge Engine (right). Workstations function as rendering engines or as display systems. Each Pixel Merge Engine (PME) connects a local frame buffer to a pair of data streams on the Pixel Bus through a user-programmable pixel combining operator.

A popular combining function for volume visualization applications is (pre-alpha multiplied) *Over* [15]. The value of a *Over* b is the result of blending a (semi-transparent) foreground pixel a over a background pixel b. It is defined

$$\begin{aligned} O(a,b) &\equiv O((r_a,g_a,b_a,\alpha_a),(r_b,g_b,b_b,\alpha_b)) \\ &= (\alpha_a r_a + (1-\alpha_a)\alpha_b r_b, \\ &\quad \alpha_a g_a + (1-\alpha_a)\alpha_b g_b, \\ &\quad \alpha_a b_a + (1-\alpha_a)\alpha_b b_b, \\ &\quad \alpha_a + (1-\alpha_a)\alpha_b)\,. \end{aligned} \tag{4}$$

O is associative [2] which implies

$$O(a, O(b,c)) = O(O(a,b),c). \tag{5}$$

3 Image Combining: Practice

Consider a simple visualization problem in computational fluid dynamics. The goal is to visualize the pressure field of a developing steady-state solution. A simple volume visualization technique is used to view the pressure field in three dimensions.

The pseudo-code of figure 2 might illustrate the inner loop of such a computation. The program performs a multigrid iteration to solve the fluid mechanics equations [9]. Every `nIterations` the loop calls `computeImage` to render an image of the local data into the local frame buffer. This continues until the residual solution error falls below a threshold of acceptability.

In order to run this code on a scalable computer system it is necessary to parallelize the routines `multigridSolver` and `evaluateResidual`. The calls to the *Viz::* member functions integrate the visualization hardware into the application. *Viz::loadCombiningFunction* configures programmable logic that executes a user-programmable combining function. *Viz::setCombiningOrder* is used to dynamically route pixel packets for correct associativity. And *Viz::combineImages* starts the image combining process.

```
Viz::loadCombiningFunction ("Over.fnc");
double residual = HUGE_VAL;
while (residual > threshold)
{
    for (int i=0;i<nIterations;++i)
        multigridSolver ();
    evaluateResidual (&residual);
    computeImage ();
  Viz::setCombiningOrder ();
  Viz::combineImages ();
}
```

Figure 2: Pseudo-code for the inner loop of a simulation with concurrent visualization. This pseudo-code is appropriate for simulation on an individual workstation, or on scalable computer systems. *Italicized text* represents calls to the visualization system application programming interface (API).

4 Hardware Realization

Figure 1 illustrates the system architecture. The architecture is realized by a set of workstations that function as rendering engines and display systems. Each workstation contains (possibly multiple) graphics accelerators and Pixel Merge Engines (PMEs) connected to a high-speed packet routing Pixel Bus. The PME primary card is a full-length 64 bit PCI card that contains programmable logic (FPGAs) and off-the-shelf ASICs. The only custom hardware in the PME is the unassembled printed circuit boards.

A PME may include daughter-cards or sister-cards to support various I/O requirements. The PME computes the pixel-by-pixel merge (1) of a pair of images. One image is normally from a local frame buffer and the second image from the Pixel Bus. In normal operation a result image is output to the Pixel Bus which routes it on to another PME in the merge sequence. The images are processed as pixel streams and are pipelined at the granularity of individual network packets. Credit-based flow control ensures that buffers do not overflow and allows distributed synchronization at the start of each frame. Data compression is used to increase the effective network bandwidth (and thus the pixel fill rate). We have experimented with a simple first-order entropy encoding that consistently produces better than 50% compression on a set of 37 test images.

Figure 3: Images of the MGF office model under direct lighting with (left) OpenGL spotlights and (right) a Monte Carlo solution of (6).

Figure 4: First generation PME prototype with PCI Pamette and Colorado-1 daughtercard. The cards are joined at the PMC connector and face each other in a single PCI slot. Cables attach the daughtercard to the Pixel Bus.

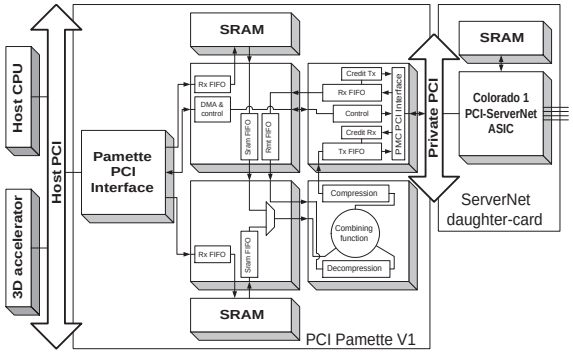


Figure 5: Layout of first generation PME prototype firmware on PCI Pamette.

All of the components required to implement this visualization architecture are available off-the-shelf with the exception of the Pixel Bus and the Pixel Merge Engine. We have chosen to implement the Pixel Bus using ServerNet-II, a low-cost gigabit network technology developed by Compaq. A single ASIC (the "Colorado-2") provides a 66 MHz 64 bit PCI interface to a pair of bi-directional 1.25 gigabit-per-second data channels and can sustain 340 megabytes-per-second of data traffic through the PCI interface. The channels are routed through 12-port non-blocking crossbar switches. The switches implement hardware channel bonding which allows them to be cascaded into scalable fat-tree topologies. These topologies guarantee constant bandwidth to all endpoints in the network irrespective of the pattern of communication and are switch-efficient at large scales [7].

The PME supports a high-speed in-band control protocol that allows one PME to modify the control state of other PMEs through the Pixel Bus. This capability is important for multi-pass algorithms in which the system has to be reconfigured quickly while rendering a single frame. It makes it possible to change the source or destination of PME image operands, modify the frame buffer configuration or pixel formats, copy frame buffer contents between graphics accelerators, and perform other tasks related to data transfer patterns and PME operating modes.

4.1 First Generation Prototypes

We have built a series of three prototype clusters (in Palo Alto, Cupertino, and Paris) using Intel and Alpha CPU workstations running Windows NT. We have experimented with several inexpensive and moderate cost PCI-attached graphics accelerators and have built clusters with boards based on the 3DLabs Permedia-2 and Evans & Sutherland RealImage 2100 chipsets.

In this generation of prototypes the PME primary card is a Compaq PCI Pamette which is a flexible FPGA prototyping platform used in a variety of system design exercises [10, 11]. Figure 4 shows the Pixel Merge Engine for one node. The large card is the Pamette and the smaller card contains the Colorado ASIC. The ASIC is a predecessor generation ("Colorado-1") that sustains 40 megabytes-per-second data transfers. The two cards are mated through a PCI Mezzanine Connector (PMC) and secured with standoff attachments. The Pamette contains five Xilinx XC4044XL packages, one of which usually supports a host PCI interface. The firmware for that interface was reused for the PCI interface to the network ASIC. Figure 5 shows how the PME firmware was partitioned on the Pamette.

We have measured the latency and frame rate of this prototype in order to extrapolate to large systems. The end-to-end latency is bounded conservatively by the sum of the latency through the PMEs and network, and the latency to merge a full frame on a single PME.

At 1024 × 1024 pixel resolution this full latency is less than $1/30$ second and is within the requirements for interactivity.

The latency through a PME varied from 6 to 13.5 microseconds per network packet, which extrapolates to no more than 13.5 milliseconds in a 1000 node system. The network adds 300 nanoseconds of latency per switch, and each path has a worst-case length of five switches or 1.5 microseconds (1.5 milliseconds at 1000 nodes). The PME produces one new pixel per clock cycle (66 megapixels-per-second in the second generation prototypes). At 1024 × 1024 pixel resolution the merge requires 15.9 milliseconds through a PME. The sum of these worst-case extrapolations is 30.9 milliseconds on 1000 nodes.

The flexibility of the Pamette has allowed us to experiment with various data transfer strategies and triggering mechanisms. We found the ultimate limiter of performance to be the achievable data transfer rate from the frame buffer. Although some chipsets include built-in DMA engines these are typically in use by the graphics driver. As a result the Pamette initiates the transactions as the DMA master. The best result we sustained was 34 megabytes-per-second with the RealImage 2100 chipset.

The data rates achieved in these experiments are not sufficient for our target fill rates of 30 or 60 Hz at 1024 × 1024 pixel resolution. These targets require 170 to 340 megabytes-per-second combined with data compression (see figure 1). In the second generation prototypes we intend to explore solutions to this bandwidth problem in the form of PCI-X and 66 MHz PCI host interfaces, and proprietary digital interfaces that are available on some graphics accelerators.

4.1.1 Images of Direct Lighting

We have experimented with speeding up a diffuse direct lighting calculation. Table 1 shows some initial results that include speedup of 6.78 on 8 computers. Figures 3, 6 and 7 show the office and conference models from the Materials and Geometry Format data base [3]. Realistic lighting for diffuse environments can be calculated by integrating the direct irradiance (6) over a set of planar lights i at every visible surface point x,

$$\Phi_i(x) = \int_A \mathcal{L}\left(x', \omega'\right) \frac{\cos\theta \cos\xi}{\|x' - x\|_2^2} dA. \tag{6}$$

This can be approximated by the OpenGL lighting equation for spotlights [14]. ξ is the difference between the angle of light emission and the surface normal vector at the point x' on the light. This corresponds to an OpenGL spotlight with exponent of 1 and cutoff of 90 degrees. θ is the angle difference between the surface normal vector at x and the vector from x to x'. This corresponds to the arccosine of the OpenGL diffuse term $L{\cdot}n$. The strength of the OpenGL diffuse light component is multiplied by the light surface area in order to incorporate the dA term. Quadratic attenuation (k_q in OpenGL) accounts for the distance term $\|x' - x\|_2^2$. k_q must be rescaled appropriately if the model has been rescaled.

The light intensities and material reflection coefficients were normalized to the range 0.0 to 1.0. Despite this precaution the colors in the resulting images are disappointing when compared to a set of reference images [5]. The differences are model-dependent (compare figures 3 and 6). We are in the early stages of understanding these and other issues related to image artifacts.

4.1.2 Scalability of Round Robin Distribution

Speedup requires effective load balancing. We experimented with the simplest of workload distribution methods, round robin partition of geometric primitives. An illustration from this experiment on four nodes appears at the start of this paper. We simplified the conference model slightly by reducing the set of 24 overhead lights to an appropriately rescaled set of 8 lights. Table 1 shows the elapsed times that were measured using 200 MHz Intel platforms with different numbers of partitions. By following a previous analysis of ray-tracing distribution [6] it is possible to analyze the scalability of any pseudo-random partitioning such as this round robin scheme. This analysis leads to the predictions in table 2.

	n	Image A	Image B	Image C	Image D	Mean
T	1	4347	4366	5647	2319	4170
	2	2184	2269	2884	1203	2135
	4	1159	1175	1484	656	1119
	8	613	650	791	375	607
S	1	1	1	1	1	1
	2	1.99	1.92	1.96	1.93	1.95
	4	3.75	3.71	3.80	3.53	3.70
	8	7.10	6.72	7.14	6.18	6.78
ϵ	1	0	0	0	0	0
	2	0.005	0.039	0.022	0.038	0.024
	4	0.067	0.077	0.051	0.132	0.073
	8	0.127	0.191	0.120	0.294	0.165

Table 1: Time T in milliseconds (worst-case among n computers); speedup $S = T(1)/T(n)$; and workload imbalance $\epsilon = nT(n)/T(1) - 1$ for conference room images A through D on up to 8 computers. See figures 6 and 7.

Pr^n	n =2	4	8	16	32	64	128
ϵ =0.1	.885	.672	.314	.045	.000	.000	.000
0.25	.986	.926	.718	.307	.025	.000	.000
0.5	.999	.993	.947	.735	.266	.009	.000

Table 2: Probability Pr^n (8) of achieving balance within ϵ by round robin distribution of the conference model with σ =500 and μ =35.553. Compare to measured ϵ in table 1.

The central limit theorem states that when a sufficiently large number of samples w_i are drawn from a sample population the sums of any sets of k samples will take on a normal distribution. If the sample population is uniform then it is possible to quantify the probability that the sum of any individual set of k samples is below a given bound y. In the general case we have

$$\begin{aligned} P\left[\sum_{i=1}^{k} w_i \le y\right] &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{(y-k\mu)/\sigma\sqrt{k}} \left(e^{-u^2/2} du\right) \\ &= \frac{1}{2} + \frac{1}{2}\mathrm{Erf}\left(\frac{y - k\mu}{\sigma\sqrt{2k}}\right). \end{aligned} \tag{7}$$

In this expression μ and σ^2 are the mean and variance of the sample population. When this is applied to the problem of rendering images in OpenGL the sample population is the set of measurements of times to render individual geometric primitives. We partition the primitives into sets of size $k = m/n$ where m is the total number of primitives in the model and n is the number of computers. We want to predict the expected imbalance ϵ for increasing n. Let T_{min} represent the ideal run time with n computers in perfect balance. From the previous definitions $T_{min} = k\mu$ and so

$$(1 + \epsilon)T_{min} - k\mu = \epsilon k\mu = \epsilon m\mu/n.$$

Use this to obtain a formula for a new quantity Pr,

$$Pr \equiv P\left[\sum_{i=1}^{k} w_i \leq (1+\epsilon)\, T_{min}\right] = \frac{1}{2} + \frac{1}{2}\mathrm{Erf}\left(\frac{\epsilon m \mu}{n\sigma\sqrt{2k}}\right). \quad (8)$$

Pr represents the probability that an individual computer will have a workload within a factor ϵ of T_{min}. The probability that all n computers will be balanced to within a factor of ϵ is the product of n instances of Pr, that is, Pr^n. This quantity can be computed for different numbers of computers.

Although it is difficult to measure the time to render an individual primitive we can infer it from the total time required to render the model. The model contained 117342 primitives and took an average of 4170 milliseconds to render. From this we obtain a value of $\mu = 35.553$ microseconds per primitive. We fit equation 8 to the measured data in table 1 to estimate the value of σ. A value of $\sigma = 500$ gave Pr^n values slightly above 0.5 of achieving the measured ϵ and appear to be in good agreement with the data.

Extrapolated predictions up to 128 nodes appear in table 2. These suggest that round robin distribution for the conference model has probability less than 0.5 of achieving 25% or better load balance on 16 nodes. These results suggest that round robin distribution of the conference model is only feasible for small clusters. These results will improve for larger models.

4.2 Second Generation Prototype

We are presently constructing a second generation prototype system based on Alpha CPU workstations running Linux. The Alpha systems have the advantages of increased memory and PCI bandwidth versus the Intel systems, as well as a more powerful CPU.

The second generation PME will consist of from one to three cards. A primary card will contain a Colorado-2 network ASIC and the FPGA packages. It will support 30 Hz operation at 1024×1024 resolution. A sister card, occupying a second PCI slot, will contain a second Colorado-2. These two cards connect by a ribbon cable to support 60 Hz operation. In addition the primary card supports a modular daughter-card through which it attaches to proprietary frame buffer interfaces.

5 Discussion

We are in the early stages of experimenting with applications on this architecture and have been exploring scientific visualization and realistic lighting calculations. In principle this visualization architecture could be supported on shared or distributed-shared memory computer systems as well as on Beowulf-class clusters. The way in which individual CPUs access their data is unimportant to the PME which is only concerned with access to a frame buffer.

It is very easy to implement an accumulation buffer pipeline in the PME and this is one way to perform anti-aliasing. This can also be useful in lighting calculations that involve shadows. An accumulation buffer implemented in this way has a lower latency than a traditional accumulation buffer, which cannot read back the data until the accumulation has completed.

This architecture incorporates emerging technology and we look forward to better OpenGL support on Alpha Linux and improved frame buffer interfaces, both of which are currently becoming available. The availability of FPGAs and inexpensive gigabit networks have been critical to this project. Our intent with this second generation is to establish a stable hardware platform that can support many generations of firmware and changing I/O requirements. We expect this architecture to evolve as we encounter new and diverse application requirements.

References

[1] James Ahrens and James Painter. Efficient sort-last rendering using compression-based image compositing. In *Proceedings of the Second Eurographics Workshop on Parallel Graphics and Visualization*, 1997.

[2] James Blinn. Compositing - theory. In *Jim Blinn's Corner*, pages 179–190. Morgan Kaufmann Publishers, 1998.

[3] Gregory Ward Larson et al. *Materials and Geometry Format*. Lawrence Berkeley Laboratory, 1999. `http://radsite.lbl.gov`.

[4] John Eyles, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, Nick England, and Lee Westover. PixelFlow: The realization. In *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 57–68, 1997.

[5] Alan Heirich and James Arvo. Scalable Monte Carlo image synthesis. *Parallel Computing*, 23(7):845–859, 1997.

[6] Alan Heirich and James Arvo. A competitive analysis of load balancing strategies for parallel ray tracing. *The Journal of Supercomputing*, 12(1/2):57–68, 1998.

[7] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34:892–901, 1985.

[8] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap image composition. *IEEE Computer Graphics and Applications*, 14(4), 1995.

[9] Stephen McCormick. *Multigrid Methods*. SIAM Press, 1987.

[10] Laurent Moll, Alan Heirich, and Mark Shand. Sepia: Scalable 3D compositing using PCI Pamette. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 1999.

[11] Laurent Moll and Mark Shand. Systems performance measurement on PCI Pamette. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 1997.

[12] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):22–32, 1994.

[13] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. InfiniteReality: a real-time graphics system. In *Proceedings of SIGGRAPH 97*, pages 293–302, 1997.

[14] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley, 1997.

[15] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of SIGGRAPH 84*, pages 253–259, 1984.

[16] Claudio Silva, Arie Kaufman, and Constantine Pavlakis. PVR: High-performance volume rendering. *IEEE Computational Science & Engineering*, 3(4):16–28, 1996.

[17] Paul H. Smith and John van Rosendale. Data and visualization corridors. Technical report, California Institute of Technology, Center for Advanced Computing Research, 1998.

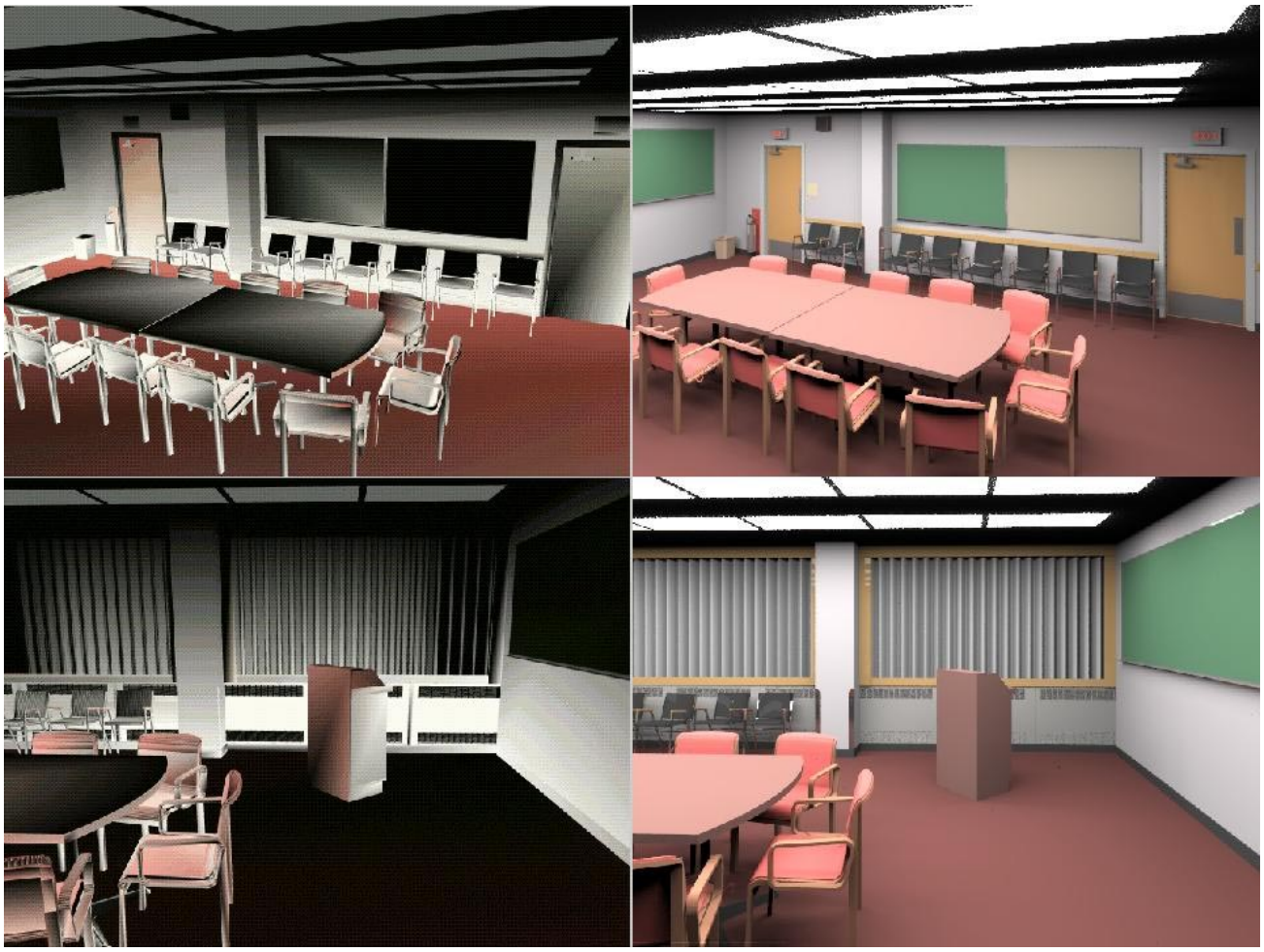


Figure 6: The MGF conference model, images A and D, rendered by OpenGL (left) and by a Monte Carlo method (right).

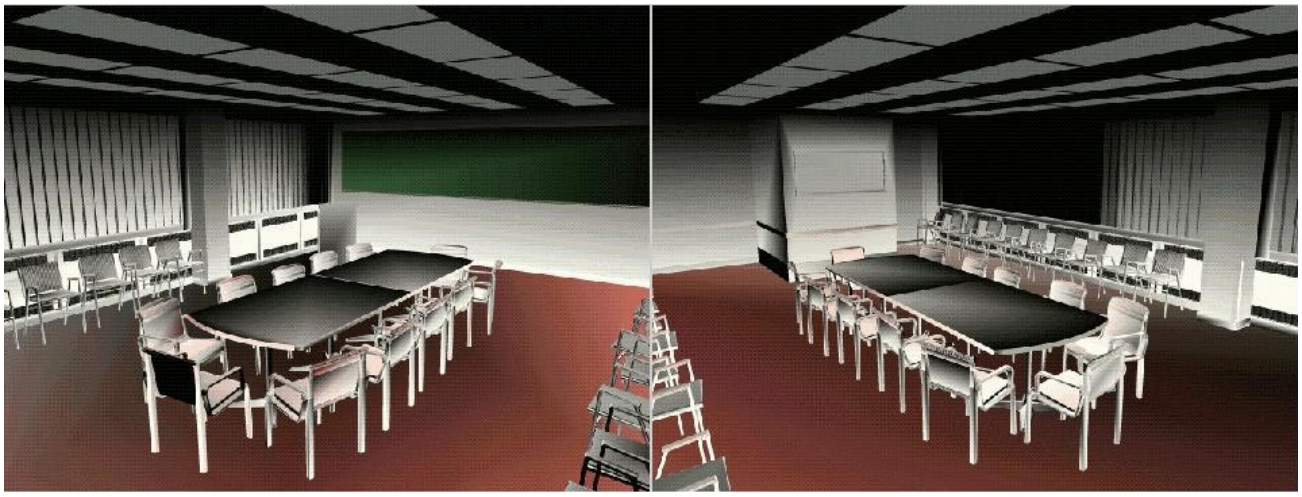


Figure 7: The MGF conference model, images B and C, rendered by OpenGL.