Hierarchically Composited Ray Cast Volume Rendering

Tom Malzbender
Systems Architecture Laboratory
HPL-90-28
April, 1990

volume rendering; compositing; computer graphics

An approach to parallelizing ray cast volume rendering on the ICE architecture is presented. ICE (Image Compute Engine) is a parallel, scalable multicomputer aimed at supporting interactive, state of the art graphics rendering. General purpose processors with high floating point performance are employed with distributed graphics hardware to achieve high performance in both graphics and general purpose computing on the same system. To ensure tight coupling between simulation and graphics ICE supports object space subdivision of graphics tasks as well as image space subdivision. Processors are grouped into clusters that render a full screen resolution image of their data set, called a *semiframe*, and a hierarchy of compositing chips are responsible for correctly combining the semiframes into a complete image. This is done without consuming interprocessor communication bandwidth. This paper describes an object space approach to volume rendering within this architecture and the hardware requirements that are placed on the compositing chips. It is found that volume rendering can be supported with simple and cheap extensions to the compositing chips.

Introduction

The capability of converting 3 dimensional arrays of data to images representing that data has come to be known as volume rendering. Within volume rendering several approaches exist, their choice dependent largely on the type and size of data that one is interested in visualizing. One of the most general approaches is that of ray casting. This is similar to ray tracing except that secondary rays due to reflections and refractions are not generated. In addition shadow rays are not calculated. Generally, the the raw data is first mapped into color and opacity values, usually implemented with lookup tables. Next a ray is generated corresponding to each pixel on the screen and as it passes through the transformed data array, color values are integrated along the ray as a function of the opacity values. High values of opacity at a particular *voxel*[1] will cause a strong color contribution from that voxel as well as attenuating color values behind that particular voxel. Transparency effects are generated with this technique.

Although this capability is becoming important for various fields such as medical imaging, fluid dynamics, electrochemisty and others, image computation time can be hours on a sequential machine. Unfortunately, there is a strong need for interactive response in volume rendering, because the visual system is incapable of perceiving all of the data in one image, and the user must experiment with various mappings, as well as viewpoints. This paper addresses accelerating ray cast volume rendering in the context of a general purpose graphics architecture that we are pursuing in the Visual Computing Department at HP Labs.

Compositing Architecture

The ICE architecture (Image Compute Engine) addresses the graphics needs of multicomputer workstations. One goal of the architecture is to allow efficient parallelization of graphics tasks among a variable, and possibly large, number of processors. Due to memory contention, the approach of global shared memory in general and a single frame buffer in particular is impractical, except for a small number of processors. This problem will become even worse in the future as memory capacities rise, allowing less potential bandwidth per Mbyte. In ICE, processors are grouped into clusters that each have their own full resolution local frame buffer. Of course, large amounts of memory are required for this approach, but as we will see the resulting graphics performance is extremely scalable. The frame buffers can be comprised of conventional dynamic ram instead of video rams and used as data or program memory when not being treated as frame memory. By 1993, memory prices should be down to roughly $10/Mbyte [McClean 88] making this approach cost effective. This approach allows efficient support of object space decomposition as well as image space decomposition of graphics tasks. Object space decomposition simply means allowing the data that is to be rendered to be subdivided in some manner. In this paper

[1] Voxel is an abbreviation for *volume element*

we simply spatially subdivide the data array, allocating a block to each processor. This capability is useful for simulation, volume data visualization, and scientific visualization in general.

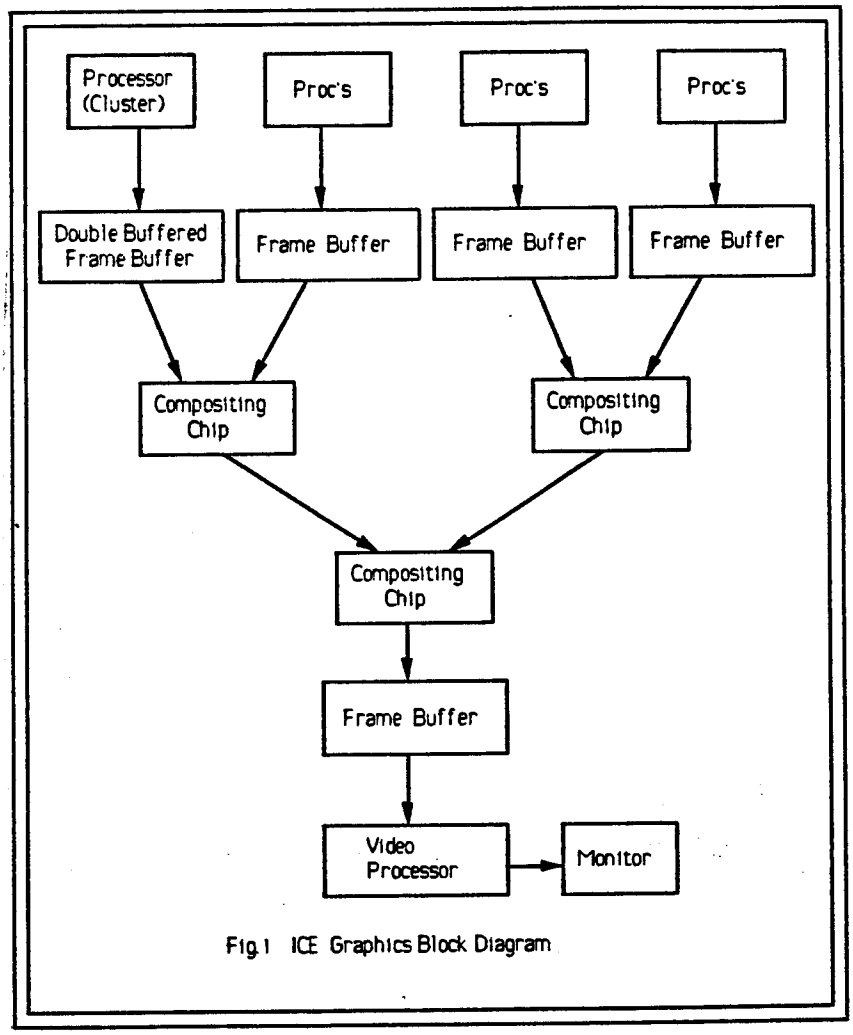

Fig.1 ICE Graphics Block Diagram

A block diagram showing only the graphics subsection of ICE is given in Figure 1. Note that the issue of processor interconnection is not addressed. At the heart of the architecture are compositing chips that are capable of combining image data from 2 sources. This is done one pixel at a time at high rates (50nsec.) These compositing chips are arranged in a hierarchy interconnecting an arbitrary number of processors generating image data into local frame buffers. The compositing chips function in various modes determined by the type of rendering that is being performed. If the image data held in the frame buffers are simply color and Z values as generated by conventional polygon rendering, compositing is a simple task. For each pixel, the Z values are compared and the source with the minimum Z value passes both Z and color values down the hierarchy. The next level of the hierarchy now performs the same operation until the root of the tree is reached. This process was first described by [Fussell 82] in which he assigned only one polygon per processor. If the compositing hardware can be made to run at video rates [Molnar 88] there is no need for an additional frame buffer at the root of the tree and instead data can be streamed out

directly to the Video look-up tables and D/A converters.

In ICE the compositing chips do not run at video speeds and results are stored at a display buffer at the root of the compositing hierarchy. This allows slower compositing times and more functionality to be implemented within the compositing chips. Each local frame buffer is double buffered and while the processors are writing graphics data into one half, the other is being accessed to feed the compositing chips. This avoids any contention for the local frame buffer memory.

Essentially linear performance is achieved over a wide range of processor counts. The depth of the hierarchy is log(number of local frame buffers) , and the latency of one level is only 50 ns. In a system of 1000 processors with local frame buffers the image suffers a latency of only .5 microseconds, an insignificant amount relative to a 60Hz. display rate. In addition, the hardware complexity scales well with the number of compositing chips. If n is the number of local frame buffers then the number of compositing chips, C, is bounded by:

$$2^{\lfloor log(n) \rfloor} - 1 \leq C \leq 2^{\lceil (log(n)+1) \rceil} - 1 \qquad (1)$$

Volume Rendering

Additional capabilities can be built into these compositing chips. [Shaw 88] has looked at providing antialiasing and is building a compositor chip that approximates [Duff 85]'s algorithm. This paper is concerned with extensions that allow ray cast volume rendering.

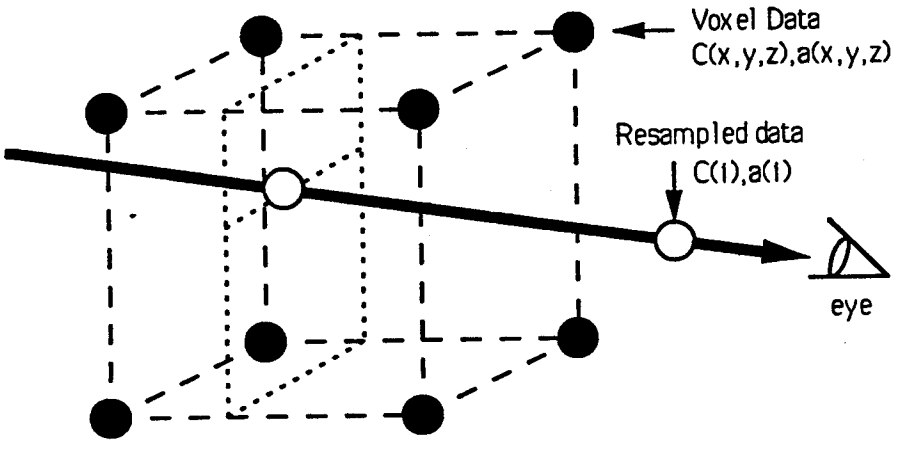

Fig. 2. Resampling data along a ray via trilinear interpolation.

In ray cast volume rendering [Levoy 88] a ray is commonly generated corresponding to each pixel on the screen, although fewer rays are generated for early stages of progressive refinement images. Figure 2 shows such a ray passing through through a 3 dimensional array of voxels that contain color and opacity values. Samples are taken at fixed intervals along each ray [Tuy 84] and nearby voxel opacity and color values are trilinearly interpolated to represent values at these sample points. More rigorous approaches to this resampling can also be taken. If the data values are considered to be point samples of a continuous field, then convolving them with a 3 dimensional sinc function to perform reconstruction is more precise from a signal processing perspective. In any case, the ray color is initialized to some background color and as the ray is traversed back to front each sample point contributes according to:

$$C_{out} = C_{in} * (1 - a_i) + C_i * a_i \tag{2}$$

where C_{in} is the ray color impinging on the sample point, C_i and a_i are respectively the colors and opacity of sample point i, and C_{out} is the color of the ray exiting the sample point. The first term simulates attenuation of the input ray while the second term corresponds to emission of color from the sample point, thereby simulating a transluscent material.

Calculation of a local gradient for each voxel is useful for two purposes. First, it can be used as a surface detector by scaling the opacity of each voxel by the magnitude of the spatial gradient of the raw data. Voxels lying at raw data transition regions will have large gradient magnitudes, yielding high opacity values that cause significant contribution to the image. The second use of voxel gradients is in the computation of shading. Here we can think of the gradient as a replacement for the surface normal, and conventional Phong or Torrance Sparrow [Cook 82] shading can be implemented. There are various operators [Horn 81], [Drebin 88] that can be used to compute the voxel gradient. One common technique [Westover 89] is the use of central differences in which six neighboring samples form an estimate.

$$\begin{aligned} grad[x,y,z] \;=\; & (data[x+1,y,z] - data[x-1,y,z]), \\ & (data[x,y+1,z] - data[x,y-1,z]), \\ & (data[x,y,z+1] - data[x,y,z-1]) \end{aligned} \tag{3}$$

Spatial Coherent Hierarchies

A goal of the ICE architecture is to provide the capability of object space subdivision. For volume rendering this means the data array is broken up and distributed into local memory of each of the processors. For large arrays of data this is an attractive approach since not enough memory is likely to exist for each processor to have its own copy of the data. Recall that a 512x512x512 array of bytes yields 128 MBytes, an excessive amount of memory to allocate for each processor in a multiprocessing system. This approach is also more attractive than allowing all processors access to one block of shared memory that holds the database. A single global shared memory will encounter difficult contention problems and fail to scale well with processor count.

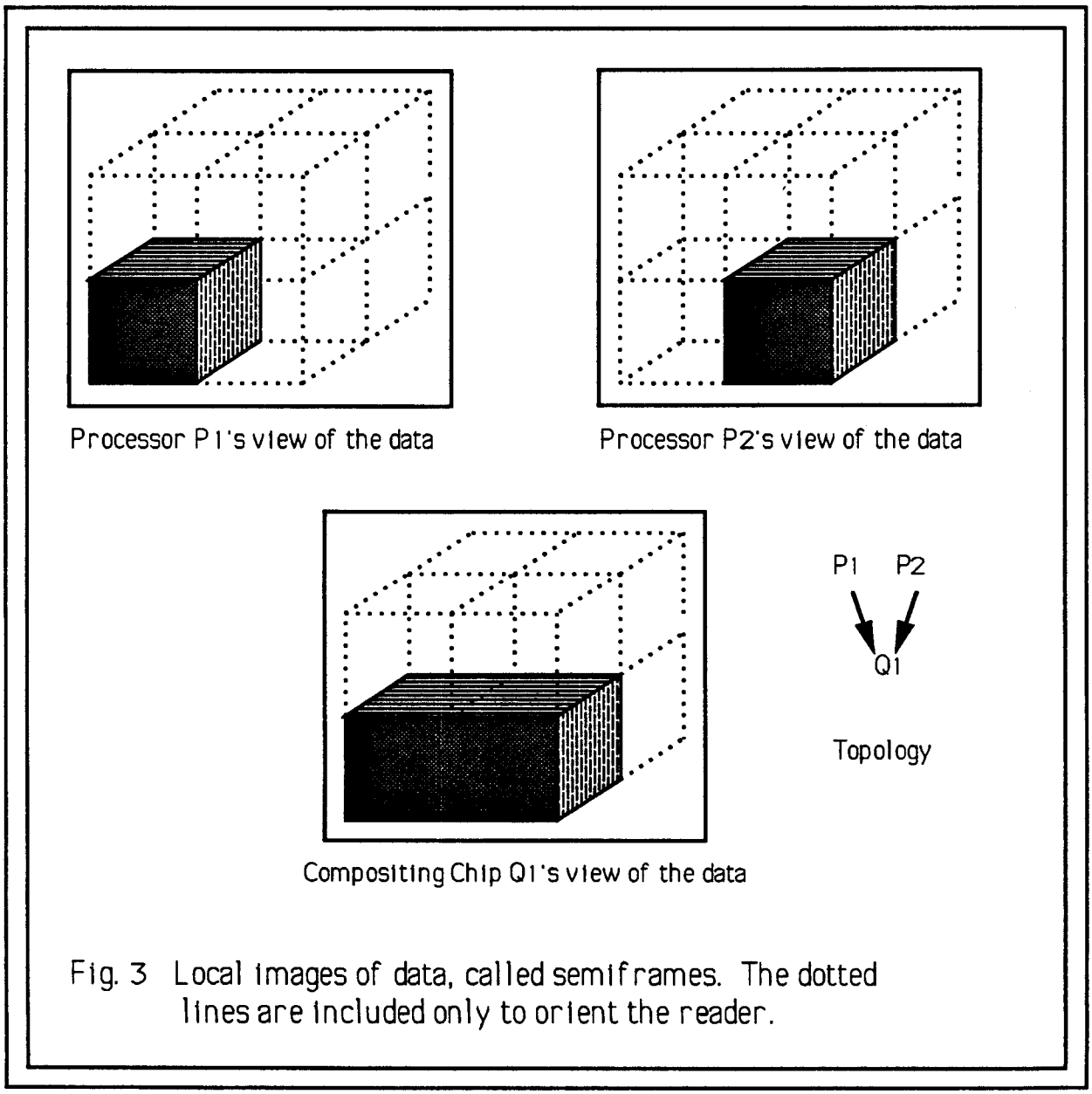

Fig. 3 Local images of data, called semiframes. The dotted lines are included only to orient the reader.

Once the data is spatially subdivided among the various processors, each one renders a full resolution image that will only include contributions from that localized data subset, as shown in figure 3. We will call such images *semiframes*[2] because they are incomplete. These semiframes are then composited hierarchically to form the final displayable color image held in the root frame buffer. As a new viewpoint is chosen, or image parameters such as opacity lookup tables or light source positions are changed, there is no need to move any raw data between the processors. Image generation commences as soon as parameter changes are broadcast to the processors.

One challenge when subdividing the data among various processors is that ray cast volume rendering requires that data samples be composited in an ordered fashion. We have described a back to front traversal along the ray, a front to back traversal is also acceptable. Given one of these two choices the data traversal order varies as the viewpoint is changed. We have attacked this problem through the use of spatially coherent hierarchies, an example of which is given in figures 4 and 5. For ease of visualization these figures and the description will limit itself to a two dimensional case, but the behavior is the same for 3 dimensions. Imagine that we have a two dimensional array of data evenly subdivided among 16 processors, P1-P16 shown in figures 4 and 5. These processors can be interconnected in a hierarchy that has the property that every node in the hierarchy merges data that is spatially neighboring. This hierarchy consists of the compositing chips labled Q,R,S and T. As one proceeds down the hierarchy, larger subspaces of data are merged , but always in a spatially coherent manner. We will see that this will allow us to composite data along arbitrary directions through the data array.

Given a view direction, each processor can now generate a full screen resolution ray cast volume rendered image that incorporates only the data block allocated to it. Since we know our compositing hierarchy maintains spatial coherence, each node of the hierarchy needs only to know which of its two inputs are in front and which is in back to be able to correctly composite these two pixel streams together. For parallel projections, this information turns out to simply be one of 3 bits (for the 3 dimensional case) which are just the sign of the x,y and z coordinates of the viewpoint described with the origin at the center of the array. These three bits can be computed by one processor and broadcast to the compositing chips in parallel before compositing is commenced. For perspective projections each compositing chip must be given its own front/back control signals.

[2]The root *semi* can mean either partial or half; we use it in the word semiframe as meaning partial.

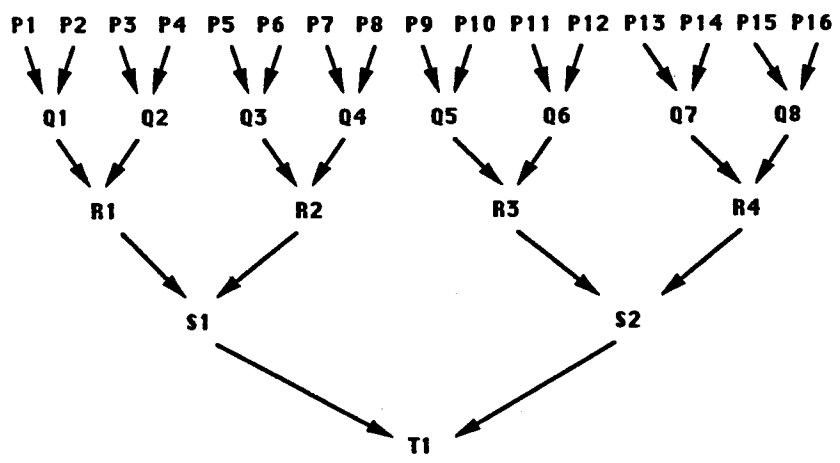

Fig. 4 Spatially Coherent Compositing Hierarchy

Topological View

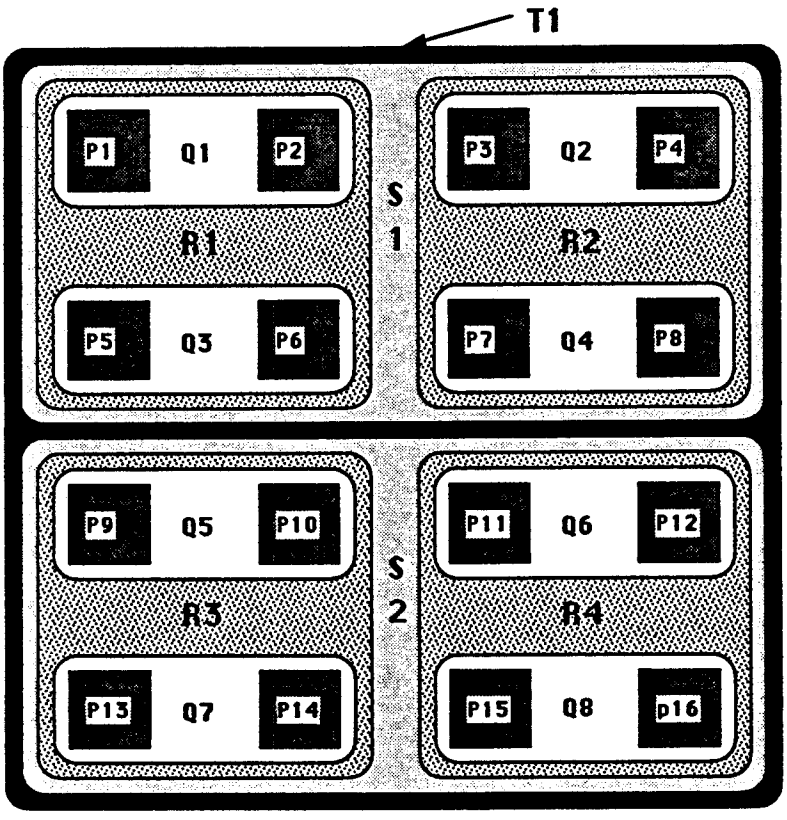

Fig. 5 Spatially Coherent Compositing Hierarchy

Spatial View

Processor Local Computations

Before any compositing is done each processor must generate a full screen resolution semiframe of the data block allocated to it. This will entail generating color and transparency values for each pixel. These transparency values will allow us to correctly composite the independent processor color bitmaps together.

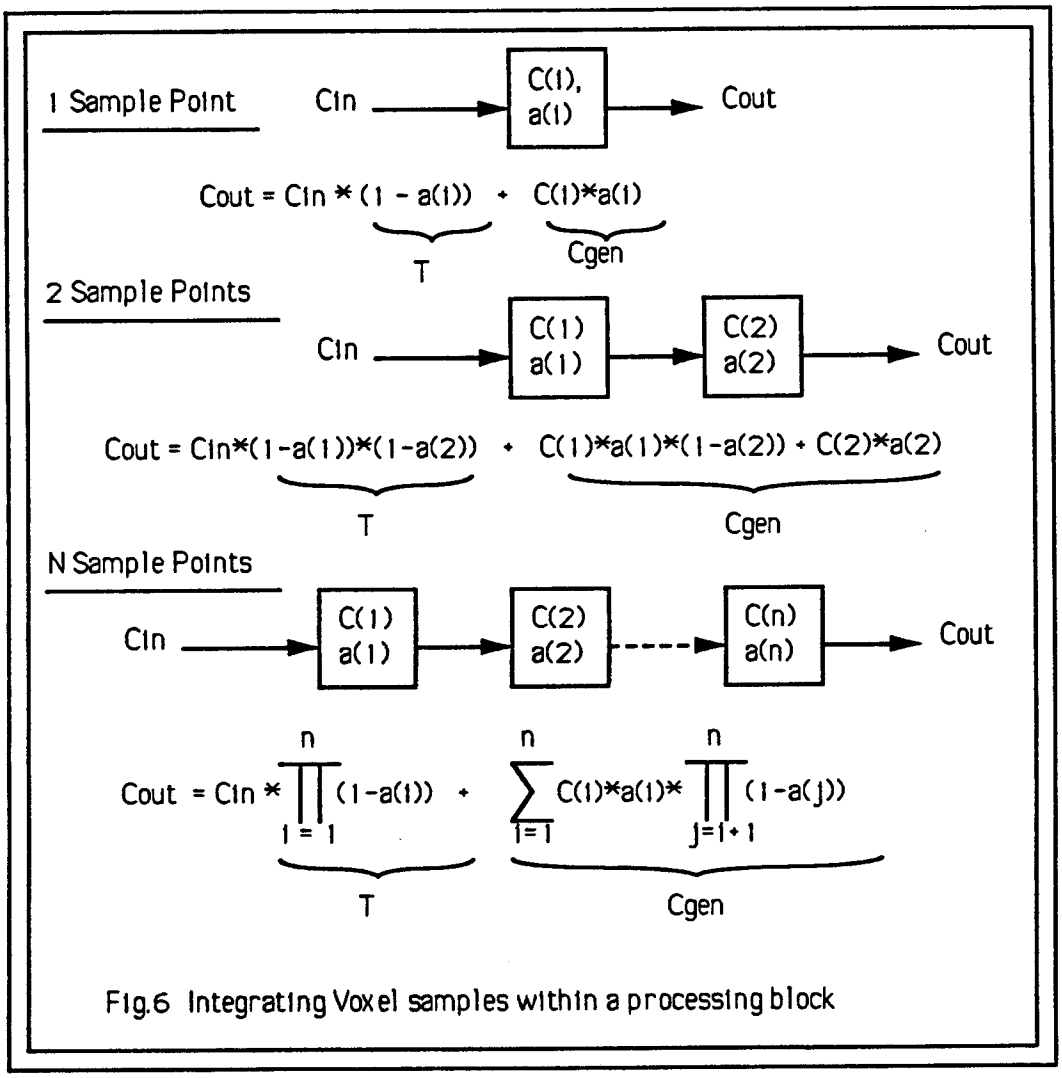

Fig.6 Integrating Voxel samples within a processing block

Figure 6 shows the case of a ray passing through 1, 2 and n sample points, each having color and opacity values that have been trilinearly interpolated from eight nearest data values. The first case, traversing 1 sample point, is simply equation 2 repeated. Color values for two sample points are easily computed since C_{out} of the first sample is just C_{in} of the second sample, allowing us to plug the equation into itself and simplify. Repeating this n times yields the equation at the bottom of the figure. Note that in all cases we have a term that represents the contribution from the attenuated input ray, $C_{in} * T$, followed by a term that represents the color contribution from the sample points under consideration, C_{gen}. One can think of T as a transparency value, and C_{gen} as a color contribution generated by a set of sample points. The effect of the n sample points encountered within a given processor can be completely described by these C_{gen} and T values, so they are stored for each pixel. The following equations summarize the computation of these values.

$$C_{gen} = \sum_{i=1}^{n} [C(i) * \alpha(i) * \prod_{j=i+1}^{n} (1 - \alpha(j))] \tag{4}$$

$$T = \prod_{i=1}^{n} (1 - \alpha(i)) \tag{5}$$

A given processor need only compute rays that pass through its data block, so the vertices of the data block can be projected onto the display and a six or four sided hull resulting used to determine which rays must be sent. Pixels outside this hull are cleared to zero C_{gen} and full transparency. Within the hull equations (4) and (5) are used. Alternately the bounding test of [Toth 85] can be used for the ray classification. Since the extent of the data allocated to each processor is known to that processor, sample points along each ray can also be restricted to where there is data.

Compositing Computations

We now turn to the task that the compositing chips have, namely to combine C_{gen} and T values from two sources into C_{gen} and T values that represents contributions from both inputs. Because the compositing hierarchy maintain spatial coherence we know that the two inputs come from data regions next to each other.

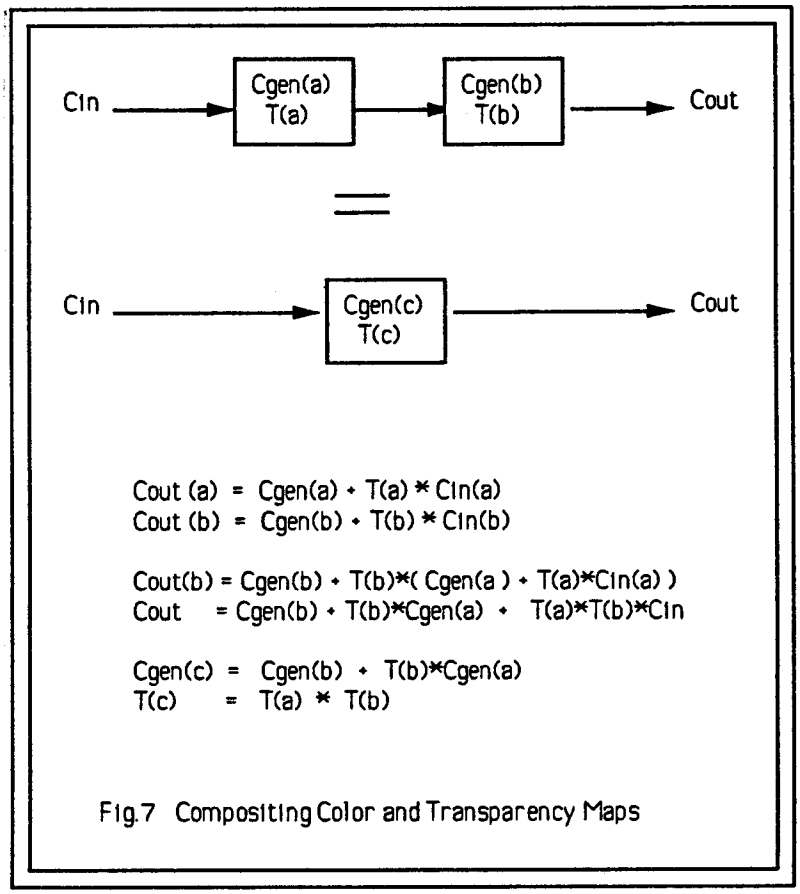

Fig.7 Compositing Color and Transparency Maps

Figure 7 considers how color and transparency values for the same pixel in two semiframes, a and b, are combined into one color and transparency value for a semiframe, c. For this example b is taken to be in front of a relative to some view direction. We start with the basic definition for how C_{gen} and T values combine for each semiframe. Since for any ray $C_{out}(a) = C_{in}(b)$ we can plug the first equation into the second. As we simplify the equations, notice that again we wind up with a term, $T(c)$ that attenuates the input ray, C_{in}, and terms that generate ray contributions, $C_{gen(c)}$. These are respectively grouped to form the transparency, $T(c)$ and color generated, $C_{gen}(c)$, values for the semiframe that represents both sources. This process is repeated at each level of the hierarchy until the complete image is formed. Although only two multiplys and one add are shown to generate these C_{gen} and T values, keep in mind that C_{gen} actually consists of a red, green and blue contribution bringing us up to 4 multiplies and 3 adds per pixel.

Hardware Implementation and Performance

The desired hardware functionality can be implemented cheaply in a gate array. The largest multipliers and adders seen in figure 8 are only 16 bit by 16 bits. However, this diagram shows only the circuitry needed for volume rendering is not a complete composing chip capable of Z-buffered pixel compositing. We will be prototyping the compositing chips with discrete components before going to a gate array implementation. A question still remains about the throughput of the compositing chips. If they can be made to clock results through at 50 ns. then a 1k x 1k image can be composited at 50 milliseconds allowing a 20 hz. display rate. Since the rendering of a 256 x 256 x 256 dataset by 64 processors is estimated to take 500msec[4] , compositing is not a bottleneck. Also, since local frame buffers are double buffered compositing can be done on one image while another is being rendered. This means that the throughput of the system is the maximum of these two times, 500msec. for this example.

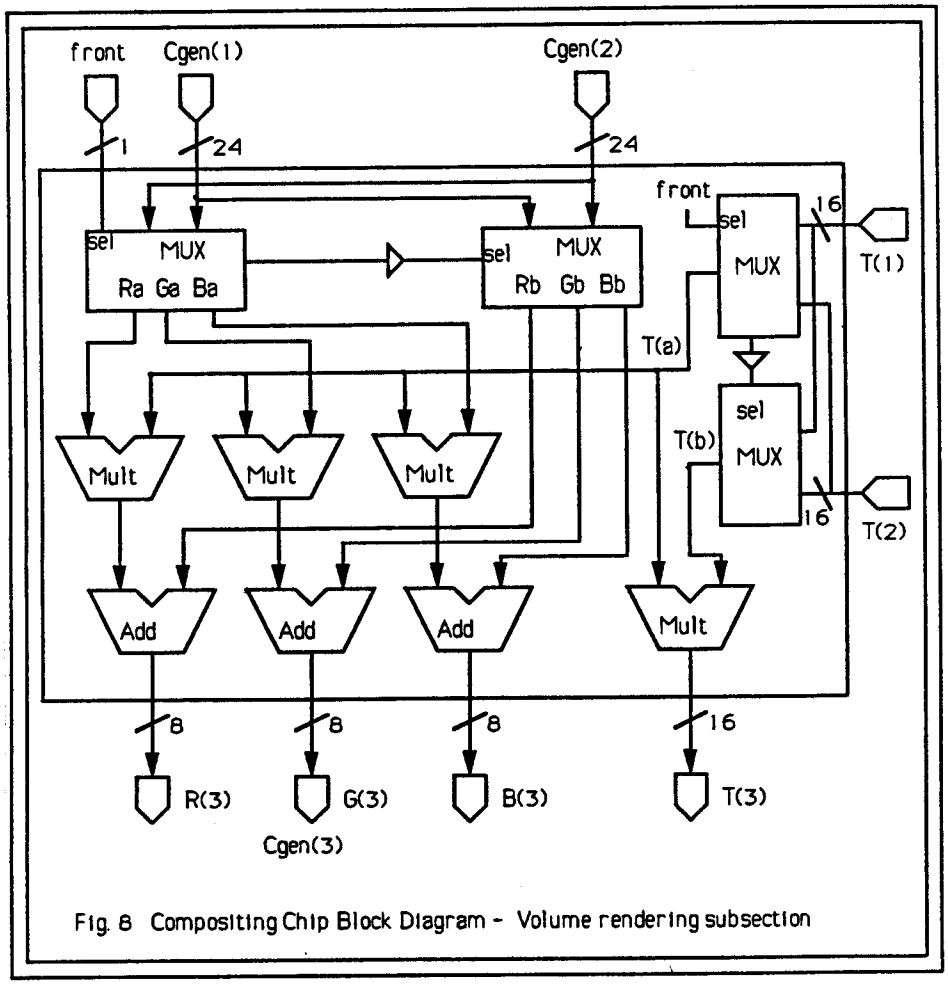

Fig. 8 Compositing Chip Block Diagram - Volume rendering subsection

[4]The 500 msec. estimate assumes 10 instructions for the trilinear interpolation, 4 for the ray accumulation, times 3 for each primary color and a 20Mflop (sustained) processor.

Tradeoff Extremes

We have made a tradeoff by performing some of the computation on general purpose processors (ray cast rendering) and some on specialized hardware (compositing). This tradeoff can be taken to extremes that are also worth addressing.

On the one hand we could perform all the compositing in software on the CPU's using no specialized hardware. In a typical case of 64 processors rendering a 256x256x256 dataset into 1k x 1k frame buffers each processor is assigned .25 Meg data samples. If we estimate 40 instructions per sample (see previous footnote), each processor needs to execute 10 million instructions. Compositing in software is still most efficiently done hierarchically, but now each level must be composited sequentially so we have the bottleneck processor executing 6 full screen compositing tasks. If we assume a compositing cost of 8 instructions per pixel, we wind up with 48 million instructions to perform compositing. This number would actually be quite a bit larger because interprocessor communication was counted as one instruction per pixel, an unrealistically fast time for a system of 64 processors. Even so, we find that our compositing time is 5 times our rendering time, so our use of specialized hardware for this operation appears warranted.

On the other hand, the compositing hierarchy could be extended until we have no processors with raw data being accessed directly by the compositing chips. A simpleminded extrapolation of the approach presented in this paper is precluded because of the brazen requirement of having a semiframe buffer at every voxel. Nonetheless, specialized VLSI distributed among raw data memory is an attractive direction for future research.

Extensions

Although this paper describes each processor performing ray cast volume rendering, the compositing chips may be used to combine transparency and color semiframes generated with other algorithms. In particular, the object space approach of [Westover 89] is appropriate. Here voxels are traversed one at a time and their contributions to the image are accumulated back to front. Lookup tables are used store 2 dimensional projections of 3 dimensional reconstruction kernels called *footprints*. Footprints are weighted by the voxels effect and accumulated to form images. If the raw data is isotropic[3], the reconstruction kernel is rotationally symmetric and images with arbitrary view directions can be computed using the same footprint lookup tables. For non-isotropic data footprint tables must be recomputed for each view. Since the technique extensively uses lookup tables for both the reconstruction and shading, it has the potential to be quite fast.

Isosurface tessellation algorithms that generate polygons such as the marchingcubes algorithm [Lorensen 87] are trivially supported in the ICE architecture, although some overlap of raw data is required between the processors. Since the compositing chips can

[3]Here *isotropic* simply means that voxels have the same spacing in each axis.

also composite semiframes consisting of color and Z values, each processor simply renders the isosurface polygons within its dataset. New isosurface thresholds or viewpoints can be broadcast to all processors and local tessellation and rendering can be commenced immediately.

The compositing chips can also be used for windowing support. If windows are allocated to processors and stored as semiframes, z values can be used to keep track of relative visibility and compositing chips used to enforce that visibility.

Conclusion

We have presented a simple and effective way of performing volume rendering in a multicomputer. Data is partitioned spatially among the general purpose processors which generate full resolution images called semiframes of their local dataset. These semiframes are combined by compositing chips arranged in a hierarchy that maintains spatial coherence. New images using different parameters, viewpoints or even algorithms can be generated without moving any of the raw data between processors. This is desirable since typical data arrays such as 512 x 512 x 512 x 1byte amount to large amounts data, 128 Megabytes in this case. The architecture is extremely scalable and the use of general purpose processors and memory leads to a high degree of flexibility.

References

[Cook 82] Cook,R.,Torrence,K. "A Reflectance Model for Computer Graphics", *ACM Transactions on Graphics*, Vol.1,No.1, Jan.1982, pp7-24.

[Drebin 88] Drebin,R.A., Carpenter,Hanrahan, "Volume Rendering", *Computer Graphics* (Proceedings of Siggraph '88), Vol.22, No.4, August 1988. pp.65-74.

[Duff 85] Duff,Tom, "Compositing 3-D Rendered Images",*Computer Graphics* (Proceedings of SIGGRAPH '85) Vol. 19, No. 3, 1985, pp.41-44.

[Fussel 82] 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.

[Goldwass 89] Goldwasser,S.,Reynolds,R.A.,Talton,D.A,Walsh,E.S., "High Performance Graphics Processors for Medical Imaging Applications",*Parallel Processing for Computer Vision and Display*, Addison-Wesley, 1989, pp.461-470.

[Horn 81] Horn, B.K.P., "Hill Shading and the Reflectance Map", *Proceedings of the IEEE*, Vol. 69, No.1, January 1981.

[Levoy 88] Levoy,Marc, "Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications*, May 1988 , pp.29 - 37.

[Lorensen 89] Lorensen,W., Cline, H., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics* (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, July 1987, pp. 163 - 169.

[McClean 88] McClean,William, *Status 1988: A Report on the Integrated Circuit Industry*, Integrated Circuit Engineering Corp., Scottsdale, Arizona, 1988, p. 7.34

[Molnar 88] Molnar, Steve, "Combining Z-buffer Engines for Higher-Speed Rendering", *Proceedings of 1988 Eurographics Hardware Workshop.*

[Potmesil 89] Potmesil,M, Hoffert,E., "The Pixel Machine: A Parallel Image Computer", *Computer Graphics* (Proceedings of SIGGRAPH '89), Vol.23, No. 3, July 1989, pp.69-78.

[Shaw 88] Shaw, Christopher D., "A VLSI Architecture for Image Composition", *Proceedings of 1988 Eurographics Hardware Workshop.*

[Toth 85] Toth, Daniel, L., "On Ray Tracing Parametric Surfaces",*Computer Graphics* (Proceedings of SIGGRAPH '85), Vol.19, No.3, July 1985, pp.171-179.

[Tuy 84] Tuy,Heang K., Tuy,Lee Tan, "Direct 2-D Display of 3-D Objects",*IEEE Computer Graphics and Applications*, October 1984, pp. 29-33.

[Westover 89] Westover, Lee, "Interactive Volume Rendering", *Proceedings of the Chapel Hill Workshop on Volume Visualization*, May 18-19, 1989, pp.51-57.