A Hardware Acceleration Method for Volumetric Ray Tracing

Lisa M. Sobierajski and Ricardo S. Avila

GE Corporate Research & Development
Schenectady, New York 12345

Abstract

In this paper we present an acceleration method for volumetric ray tracing which utilizes standard graphics hardware without compromising image accuracy. The graphics hardware is employed to identify those segments of each ray that could possibly contribute to the final image. A volumetric ray tracing algorithm is then used to compute the final image, traversing only the identified segments of the rays. This technique can be used to render volumetric isosurfaces as well as translucent volumes. In addition, this method can accelerate the traversal of shadow rays when performing recursive ray tracing.

1. Introduction

The ability to render large volumetric data sets quickly is an essential requirement in many engineering and scientific applications. This is a challenging demand, since the scientific user is typically unwilling to sacrifice the accuracy of the final image in order to decrease projection times.

One way to improve rendering performance is to take advantage of the graphics hardware supported by most workstations. Current workstations typically offer hardware methods for rendering geometric primitives, with depth buffering employed for hidden surface removal. This hardware can be used directly to render a set of geometric primitives that approximate the volume. Although there are many rendering algorithms that directly employ graphics hardware, none of these methods can be used to generate accurate images of both volumetric isosurfaces and translucent volumetric data. The marching cubes technique [8] can be used effectively to render volumetric isosurfaces, but cannot adequately represent an amorphous object. A cell projection method [13] or a polygonal splatting technique [6] can be used to capture translucent volumetric data, but the resulting images are often unacceptable when sharp details are desired.

In contrast to hardware projection algorithms, a volumetric ray tracing algorithm is generally slower, but much more accurate and flexible [9]. For example, the value returned by a ray-volume intersection may represent an isosurface intersection location, an accumulated color and opacity value, or the maximum value encountered along the ray. In addition, a ray tracing algorithm can be used to include global effects such as shadows and reflections in the final image.

The rendering method presented in this paper employs graphics hardware to accelerate volumetric ray tracing. An approximation of the volumetric data is projected using the graphics hardware, and the information stored in the color and depth buffers is used to reduce the amount of time required for a ray-volume intersection calculation. These rendering improvements are obtained by avoiding intersection calculations in regions of the volume that could not contribute to the final image. A classification of volume regions is given in Section 2. An algorithm that accelerates ray tracing with a depth buffer projection is described in Section 3. In Section 4, a color buffer version of this algorithm that improves rendering times for translucent projections is presented. These acceleration methods can also be used to reduce the time required to cast shadow rays in a ray tracing algorithm, as described in Section 5. The results of this technique are given in Section 6. Finally, some conclusions and future work are discussed in Section 7.

2. Cell Classification

A volume is a 3D rectilinear array of scalar values that define some property, such as density or temperature, at discrete grid locations. An interpolation function is employed to define scalar values between grid locations in order to produce a scalar field. We have chosen to use trilinear interpolation for the work presented in this paper. However, the acceleration algorithms readily extend to handle other interpolation functions such as zero-order (nearest-neighbor) interpolation or tricubic interpolation.

Consider the example ray in Figure 1 where a ray is cast through a volume and ray-isosurface intersection calculations are being performed along the ray. The cells encountered along the ray are shown as small cubes where the data samples that the define the scalar field within the

(See color plates, page CP-5)

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

cube are located at the eight vertices. The isosurface threshold value can be used to classify all cells in the volume as either "empty" cells, or "possibly contributing" cells. An empty cell is one where all eight vertices are either above or below the isosurface value, and therefore the surface does not pass through this cell. A possibly contributing cell is one that does contain the surface. For a specific ray, as shown in Figure 1, the possibly contributing cells can be further classified as either "non-contributing" or "contributing". The non-contributing cells, shown in light grey, indicate that a surface does pass through the cell, but the ray does not actually intersect that surface. The contributing cells, shown in dark grey, contain a surface that the ray does intersect.

These cell classifications are also valid for volume rendering techniques where, for example, an accumulated color and opacity value is calculated instead of an isosurface intersection value. In this case, the empty cells contain only scalar values with zero opacity according to the opacity transfer function, while the possibly contributing cells contain scalar values with non-zero opacity. For a given ray, the contributing cells are those in which the ray encounters scalar values with non-zero opacity. The non-contributing cells are those which contain non-zero opacity values, but the ray does not encounter these values.

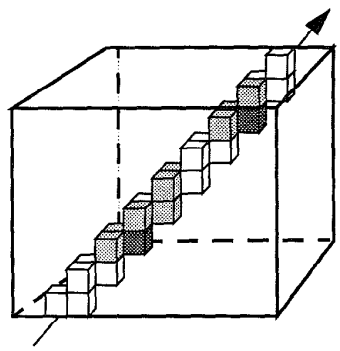

Figure 1: The classification of cells along a ray as either contributing ( ), non-contributing ( ), or empty ( ).

The standard ray casting method considers all cells along a ray when searching for an isosurface intersection or accumulating color and opacity. The ideal ray caster would look only at the contributing cells to determine the final ray value. Unfortunately, the determination of whether a cell is contributing is dependent on the scalar values, the isosurface threshold value or opacity transfer functions, and the ray. Since the ray is typically different for every ray cast, it is difficult to know for a given ray which cells are contributing.

Alternatively, the determination of whether a cell is possibly contributing is independent of the ray direction. Considering only the possibly contributing cells is equivalent to skipping the empty cells. There are various algorithms for quickly stepping through the empty cells. A hierarchical representation of the data could be constructed [6], and a ray traversal algorithm could be used to step through the cells at various levels. Unfortunately, for noisy data such as that acquired from confocal microscopy, the time required to move between levels in the hierarchy is sometimes greater than the time saved by the larger steps taken at the higher levels of the hierarchy. Another possibility is to construct a distance volume, where each cell contains the distance to the nearest possibly contributing cell [10, 14]. This method requires a significant amount of additional memory to store distance values, and the time required to build the distance volume may be prohibitively slow for interactively modifying the isosurface threshold value or opacity transfer functions. Algorithms that employ an object-order technique [4, 6] can compress the volume to remove empty cells, but cannot easily produce an accurate rendering of multiple overlapping volumes.

The goal of the work presented in this paper was to develop an algorithm that reduces the time required to cast a ray by avoiding empty cells. This algorithm can render volumetric isosurfaces as well as translucent volume data with no loss in image quality over standard ray casting. In addition, multiple overlapping volumes can be easily rendered.

3. Depth Buffer PARC

In *Polygon Assisted Ray Casting*, known as PARC, graphics hardware is used to determine the distance to the closest and farthest possibly contributing cell along each ray [1]. These distance values can be quickly obtained by projecting a simple polygonal model of the possibly contributing cells into standard zbuffer hardware.

The polygonal model consists of the rectangular polygons representing the "outer" faces of the possibly contributing cells. The outer faces are those that are shared between a possibly contributing cell and an empty cell. The polygonal model is projected first into a "near" zbuffer using the standard "less than" operator to capture the closest distance information, and then into a "far" zbuffer using a "greater than" operator to capture the farthest distance information. If a greater than operator is unavailable for zbuffering, the viewing matrix can be modified to produce inverted depth values.

Once the nearest and farthest distances have been computed for a ray, a cell stepping algorithm is used to evaluate the contribution of each cell between the bounding distances, as in the 2D example of an isosurface intersection shown in Figure 2.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

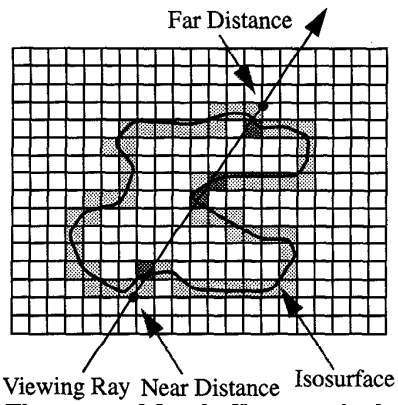

Figure 2: The near and far zbuffers contain the closest and farthest distance to a possibly contributing cell along a viewing ray.

Isosurface intersection calculations are done on a cell-by-cell basis along the ray, while samples are taken at uniform intervals along the ray for accumulation techniques. Early ray termination is possible, for example, when an isosurface intersection is detected, or the opacity along the ray reaches unity. If early ray termination does not occur, ray traversal ends at the far distance. In the special case where the eye point is located within a possibly contributing cell, the stepping algorithm must start from the viewing plane and can terminate at the farthest distance. The PARC ray tracing method is summarized in the algorithm shown in Figure 3.

- Create a set of near and far buffers for each visible volume
- For each pixel in the image do:
  - For each visible volume do:
    - Obtain near and far distances for the ray from the set of buffers for that volume
    - If near value is less than far value:
      - Cast the ray segment
  - If any contributing cells are encountered along any ray segments:
    - Shade the ray segments to get pixel color

Figure 3: The PARC ray tracing algorithm.

Decoupling the ray casting and the shading processes allows for multiple overlapping volumes, even if the volumes require different ray casting functions [9]. For example, a volumetric isosurface could intersect with a translucent volume. A standard ray tracing illumination equation is used for surface intersections [12], while a transport theory model is employed for rendering translucent data [3].

For large volumetric data sets, the number of geometric primitives in the simple polygon model often produces high hardware projection times that eliminate the savings gained during ray stepping. To reduce the number of polygons in the model, an $m \times n \times p$ group of neighboring cells can be considered one "supercell". The interpolation function can be employed to generate supercells with non-integer m, n, and p. A supercell is possibly contributing if it contains any possibly contributing cells, otherwise it is considered to be empty. The polygonal model is then created for the possibly contributing supercells, resulting in fewer geometric primitives, but generally longer ray segments between the near and far distances. Figure 4 shows the same example as in Figure 2, except that a supercell size of 3×2 was employed.

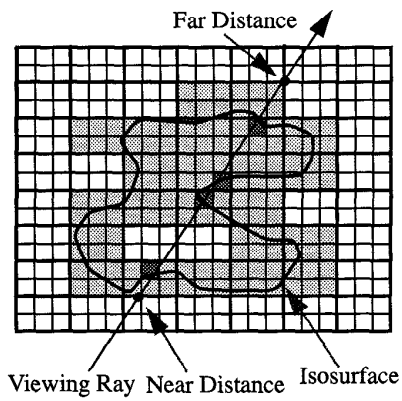

Figure 4: A 3×2 supercell is used to determine the near and far distances, resulting in less geometric primitives but longer ray segments than the example shown in Figure 2.

For small volumetric data sets, "subcells" can be created by sub-sampling the volume. This results in more geometric primitives and generally shorter ray segments. In general, the closer the polygonal model comes to accurately representing the possibly contributing regions of the volume, the higher the ratio of contributing to non-contributing segments along the ray, as can be noted by comparing Figure 2 with Figure 4. Therefore a reduction in ray casting speed can typically be obtained at the cost of higher polygon projection times. The optimal number of geometric primitives mainly depends on the relative performance of hardware polygon projection versus software cell processing.

In theory, the PARC algorithm as described above will produce images identical to those produced by standard ray casting. In practice, allowances must be made for the inaccuracies found in hardware polygon scan-conversion to achieve the identical image. This can be accomplished by small changes to the position and size of the polygon to ensure that:

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

$Z_s \leq Z_i$ for all near buffer pixels, and

$Z_s \geq Z_i$ for all far buffer pixels,

where Z_s is the scan-converted depth value, and Z_i is the ideal depth value for a pixel.

4. Color Buffer PARC

The depth buffer PARC algorithm described in the previous section can be used to skip over the initial empty cells when performing volume rendering. This technique works well when rendering a volumetric isosurface, since the contributing cell with the actual intersection location is typically encountered at, or shortly after, the first possibly contributing cell. However, when performing a color and opacity accumulation method along the ray, the opacity does not typically reach unity at the first contributing cell. In fact, all contributing cells along the ray are often considered when computing the final ray accumulation value. This is also true of other ray-object intersection functions such as maximum value and average value calculations. Therefore, volumetric ray tracing would often benefit from the ability to skip over the empty cells that occur between the first and last possibly contributing cell. This capability can be achieved by making a modification to the PARC algorithm.

Instead of projecting the outer faces of the possibly contributing cells into a zbuffer, a polygonal model for the possibly contributing cells is projected into the color buffer. Rendering is performed so that the resulting bits for each pixel in the color buffer represent segments along the ray passing through that pixel. The value of each bit determines whether the corresponding ray segment encounters any possibly contributing cells. Using this information, intersection or sampling calculations can be avoided on the empty segments of the ray.

In order to obtain this color buffer, the possibly contributing cells are assigned color values according to their distance along the major viewing axis, as shown in Figure 5. The major viewing axis is the positive or negative axis of the volume that is most closely aligned with the viewing direction. For each plane i, $0 \leq i < N$, along the major viewing direction, the color value of the possibly contributing cells in that plane is 2^i. To reduce the number of color values required, the first plane along the major viewing direction that contains at least one possibly contributing cell is considered plane 0.

The outer faces of the possibly contributing cells, and the faces that are shared by two possibly contributing cells of different color values, are projected without shading or depth buffering according to the currently defined viewing

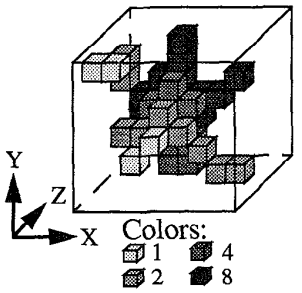

Figure 5: The color of a cell is determined by its distance along the major direction. The example colors given here are valid when Z is the major axis.

matrix. During projection, a bitwise OR operation is employed on the value currently stored in a pixel and the value being written to the pixel to determine the final pixel value. Only P planes of cells can be projected at once, where P is the number of color bits available for display on a workstation. After each set of P planes has been projected, the bits from the color buffer are saved. When all cells have been projected, the bits that are set in each pixel indicate the possibly contributing cells encountered along the ray that passes through that pixel, as shown in Figure 6.

Each bit in a color buffer pixel represents the segment of the ray passing through the corresponding plane of cells. The ray may pass through either one or two cells in a plane, and the corresponding bit is set to 1 if either cell is possibly contributing, otherwise the bit is set to 0. The ray stepping algorithm is modified to avoid sampling the planes that have a bit value of 0. Additional speedups are achieved by considering the pixel value one byte at a time, and examining the actual bits only for non-zero bytes.

When supercells are projected to obtain the color buffer values, this method is analogous to a hierarchical method with three levels. The lowest level is the original data, the middle level is represented by the pixel bits, and the highest level is represented by the bytes. This hardware acceleration method provides an approximate list of elements pierced by the ray at the two higher levels of the hierarchy. The list is approximate since two pierced supercells in one plane are represented by one value in the list. The time spent acquiring samples in empty cells in the traversal list is overshadowed by the time saved due to the hardware generation of the list.

As in the depth buffer PARC algorithm, special considerations must be made when the eye point is inside a possibly contributing cell. In this case, each viewing ray must start at the viewing plane. After the ray steps beyond the plane containing this initial cell, the algorithm utilizes the projected color bits to step along the ray as before.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

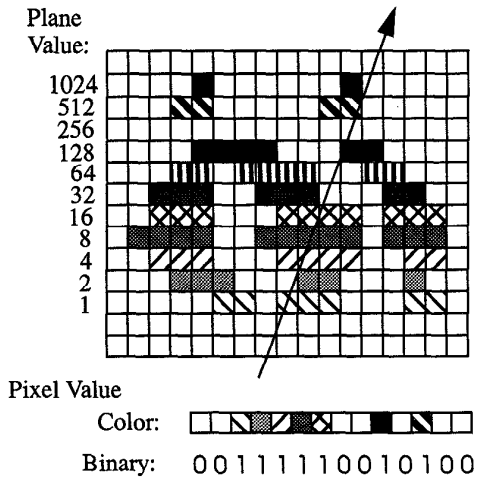

Figure 6: The color buffer values for a pixel represent the possibly contributing cells encountered along the ray passing through that pixel.

5. Shadow Rays

The depth buffer PARC method described in Section 3, and the color buffer version described in Section 4 can also be employed to accelerate shadow rays in a volumetric ray tracing method. For each point light source in the scene, the light position is treated as an eye point, and either a color buffer, or near and far zbuffers are created for each object using a perspective viewing matrix. The viewing direction is defined by the vector from the light source to the center of the object. Eight vectors are defined that point from the light source to the eight vertices of the bounding box of the object, and the field of view for the perspective projection is twice the maximum angle between the viewing ray and each of the eight vectors. If a field of view is computed to be greater than 90 degrees, then multiple projections are taken, each with a field of view that is less than or equal to 90 degrees. In the worst case the light source is located within the object, and six projections are required. Figure 7 shows an example of the buffers created for a point light source and an object using depth buffer PARC. For each directional light source in the scene, the light direction is considered the viewing direction, and a parallel viewing matrix is employed to generate the PARC buffers for each object. The position, width, and height of the viewing plane used for rendering is determined by the projected locations of the eight bounding vertices of the object onto the viewing plane.

In this shadow ray method, the width and height in pixels of the PARC buffers is arbitrary. All buffers could be set to a constant size, or the size of the buffers could vary depending upon the size of the object and distance to the light source.

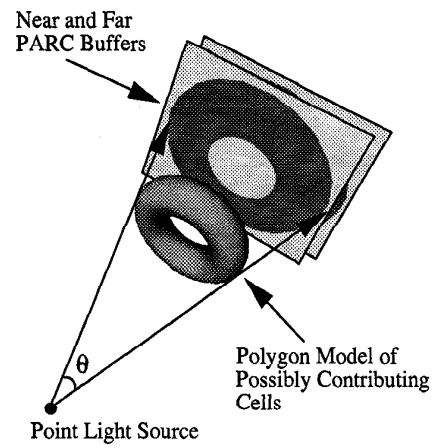

Figure 7: Near and far PARC buffers are created by viewing the volume from the light source.

When a shadow ray is cast to a light during ray tracing, the information stored in the PARC buffers for that light is employed in a manner similar to the PARC buffers for primary rays. The difference is that a shadow ray is not necessarily represented exactly by a pixel in the light PARC buffers. Therefore, the intersection of the ray with each of the PARC buffers for that light is determined. For each intersection, the values stored at the four neighboring pixels are merged to form one PARC value. For depth buffer PARC, this requires selecting the minimum near value, and the maximum far value found in the four pixels. For color buffer PARC, this involves applying a bitwise OR operation to the values found in the four pixels. In addition, the values found in the PARC buffers for a light source must be inverted to account for the fact that the shadow rays are cast in the opposite direction of the viewing rays used to create the buffers.

6. Results

The algorithms described in this paper were implemented within the VolVis volume visualization system [1, 2]. Great care was taken to ensure that these algorithms support multiple overlapping volumes, perspective and parallel projections, analytic intersection calculations, and many other VolVis capabilities. As a result, some of the possible rendering speedups have been sacrificed in order to retain high functionality.

The results were obtained on a Silicon Graphics Indigo^2 Extreme with a 200MHz R4400 processor, 24 bit color buffer, 24 bit zbuffer, and 64MB of RAM. Since a double buffered visual was employed for the rendering window, color buffer information was captured eight bits at a time. The timing information is in seconds of wall time.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

Figure 8: A volume rendered image of an LGN cell using a color and opacity accumulation method.

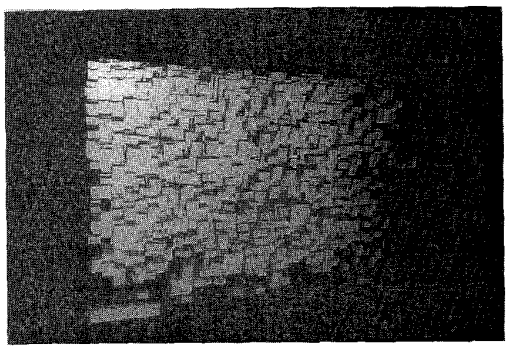

Figure 9: A polygonal model used to obtain depth buffers for the LGN cell.

Figure 8 shows an example data set where color buffer PARC performs better than depth buffer PARC for images generated using a color and opacity accumulation method. This lateral geniculate nucleus (LGN) data was obtained at a resolution of 384x256x195 using a confocal microscope. After applying the opacity transfer function, 93.5% of the cells were empty. The standard ray tracing method required 110.33 seconds to generate a 300x300 image. The depth buffer PARC method reduced this time to 60.64 seconds. The polygonal model used to generate the depth buffers is shown in Figure 9. This model contains approximately 38,000 polygons. The color buffer PARC method reduced the image time to 23.88 seconds, using a color buffer with 8 bytes. The color buffer shown in Figure 10 is 3 bytes deep for illustration purposes. The additional reduction in rendering time over the depth buffer PARC method was gained by skipping empty segments between samples.

Statistics for the images shown in Figures 11-13 are given in Tables 1-3. The image represented by the data in the fourth row of each table is the image shown in the corresponding Figure, although except for pixel size, all four images in each table are identical. Statistics for the data sets are given in Table 4. This information includes the number of data samples and the size of the volume in units. Also given are the size of the supercells in data samples, and the percentage of supercells that were classified as possibly contributing.

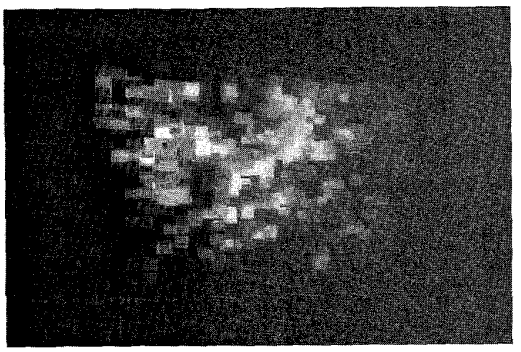

Figure 10: A color buffer used to accelerate volume rendering of the LGN cell.

Figure 11 contains a CT scan of a frozen human foot from the Visible Human Project [5] which was rendered using an opacity and color accumulation method. In Table 1, the PARC version is given in the first column, where "None" indicates standard ray tracing. The image size is indicated in the second column. Two different image resolutions were used to show that both standard ray tracing and PARC scale according to image size. The third column contains the number of samples taken along all rays, while the fourth column indicates the time to take these samples. This time includes the time required to project the polygonal model for the PARC methods. Samples were takes every 0.25 units, where the unit size of the data set was 144x247.9x220, as shown in Table 4. Finally, the fifth column indicates the total image generation time. This is essentially the ray casting time plus the ray shading time. The color buffer PARC algorithm performed almost 9 times better than the standard ray tracing algorithm.

Figure 12 contains two data sets obtained from simulation, representing the positive and negative wave function values in a high potential iron protein. The positive wave function values are rendered as a volumetric isosurface while the negative wave function values are rendered using a color and opacity accumulation method. The depth buffer PARC method was used to accelerate the isosurface, while the color buffer PARC method was used to accelerate the translucent data. The statistics for this image are shown in Table 2, where the first two columns again indicate the PARC method and the image resolution. The third column indicates the number of cells that were

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

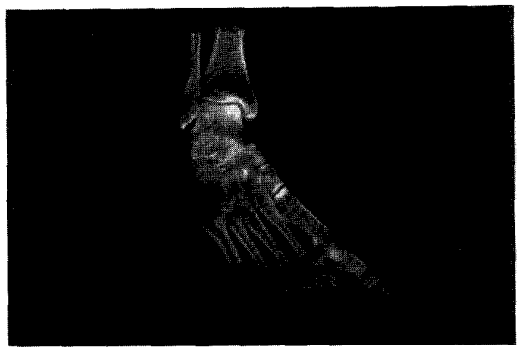

Figure 11: A translucent rendering of a human foot.

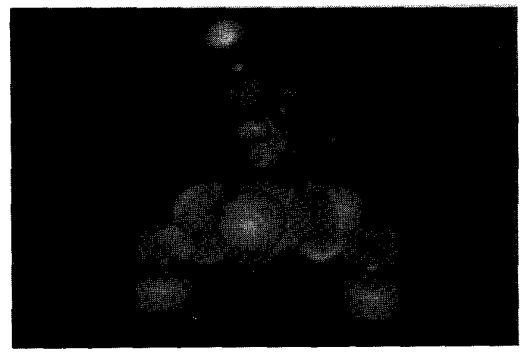

Figure 12: A combined isosurface and translucent rendering of a high potential iron protein.

considered for intersection calculations, while the fourth column contains the number of samples taken during the accumulation method. Again, samples were takes at 0.25 unit intervals. The fifth column contains the time required to cast all rays, while the sixth column indicates the total time for a shaded image. In this case, the combined color and depth buffer PARC performed more than 5.6 times faster than the standard ray tracing method.

Figure 13 shows a hippocampal pyramidal neuron casting two shadows on a volumetric floor. The cell was obtained using confocal microscopy, while the floor was voxelized from a geometric description [11]. Depth buffer PARC was used to accelerate both the primary rays and the shadow rays. The statistics for the image are given in Table 3, where the first two columns indicate PARC version and image size. The third column contains the number of cells considered for intersection during primary ray calculations, the fourth column indicates the time required to cast the primary rays, and the fifth column shows the time required to generate a shaded image with only primary rays. In the sixth column, the number of cells considered for intersection during shadow ray casting is given, while the seventh column indicates the time required for the shaded image with shadows. For primary rays, a 3.5 times increase in rendering performance is obtained, while shadow ray generation times are improved by about a factor of 2.

Table 1: Results for Figure 11

PARC Version	Image Size (pixels^2)	Samples (x 10^3)	Casting Time (s)	Total Time (s)
None	300	47250	149.12	168.43
Color	300	3318	12.70	19.16
None	600	189741	598.50	673.95
Color	600	13285	49.70	75.90

7. Conclusions

We have developed two projection algorithms that utilize standard graphics hardware to significantly reduce volumetric rendering times. The depth buffer version of PARC is well suited for rendering volumetric isosurfaces whereas the color buffer version of PARC is better suited for volume rendering. Both algorithms achieve high speedups without compromising image accuracy. The algorithms are general, supporting both parallel and perspective projections. In addition, the algorithms can be used to accelerate shadow rays.

In the future, we intend to investigate several enhancements to these algorithms. Requiring all elements (cells, supercells, or subcells) to have the same voxel dimensions leads to a compact data structure for storing the elements, but may not lead to an optimal polygonal model. We are investigating the ability to group variously sized regions of the volume in order to more closely approximate the possibly contributing cells with fewer polygons. In the color buffer PARC method, we are considering projecting several resolutions of polygonal models, enabling us to perform a hierarchical ray traversal. Finally, we are exploring the possibility of projecting additional information for each possibly contributing cell. For example, we could also indicate if a cell is homogeneous which may accelerate the rendering of translucent data.

Table 2: Results for Figure 12

PARC Version	Image Size (pixels^2)	Cells (x 10^3)	Samples (x 10^3)	Casting Time (s)	Total Time (s)
None	300	5123	15094	61.01	68.95
Color/Depth	300	183	881	8.19	12.12
None	600	20508	60436	242.92	276.02
Color/Depth	600	735	3341	31.99	43.55

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

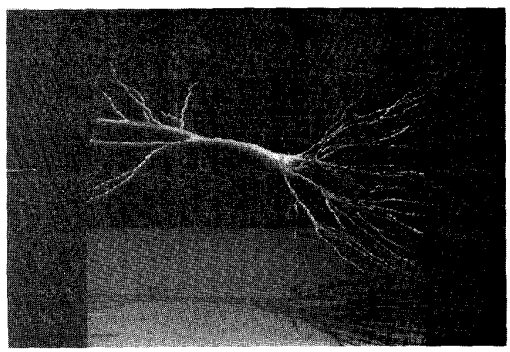

Figure 13: A ray traced image of a hippocampal cell casting two shadows on a volumetric floor.

Acknowledgements

The nerve cell data sets shown in Figures 8 and 13 are courtesy of Barry Burbach at the Howard Hughes Medical Institute, Stony Brook, NY. The CT data for Figure 11 is courtesy of the National Library of Medicine's Visible Human Project. The high potential iron protein data set shown in Figure 12 is courtesy of Scripps Clinic, La Jolla, CA.

References

1. R.S. Avila, L.M. Sobierajski, and A.E. Kaufman, "Towards a Comprehensive Volume Visualization System," *Visualization '92 Proceedings*, pp. 13-20 (October 1992).
2. R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang, "VolVis: A Diversified Volume Visualization System," *Visualization '94 Proceedings*, pp. 31-38 (October 1994).
3. W. Krueger, "The Application of Transport Theory to Visualization of 3D Scalar Data Fields," *Computers in Physics*, pp. 397-406 (July/August 1991).
4. P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation", *Computer Graphics (Proc. SIGGRAPH)*, pp. 451-457 (July 1994).

Table 3: Results for Figure 13

PARC Version	Image Size (pixels^2)	Primary Cells (x 10^3)	Primary Casting (s)	Primary Total (s)	Shadow Cells (x 10^3)	Total Time (s)
None	300	14042	33.38	35.61	12552	66.04
Depth/ Shadow	300	1490	7.72	10.06	1866	26.31
None	600	64244	132.04	141.29	49963	260.45
Depth/ Shadow	600	5999	29.72	40.29	6673	99.74

5. W. Lorensen, "Marching Through the Visible Man", *Visualization '95 Proceedings*, (October 1995).
6. D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering", Computer Graphics (Proc. SIGGRAPH) 25(4), pp. 285-288 (July 1991).
7. M. Levoy, "Display of Surfaces from Volume Data", *IEEE Computer Graphics & Applications*, **8**(3) pp. 29-37 (May 1988).
8. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics (Proc. SIGGRAPH)* **21**(4) pp. 163-169 (July 1987).
9. L.M. Sobierajski and A.E. Kaufman, "Volumetric Ray Tracing," *1994 Symposium on Volume Visualization*, pp. 11-18 (October 1994).
10. M. Sramek, "Fast Surface Rendering from Raster Data by Voxel Traversal Using Chessboard Distance", *Visualization '94 Proceedings*, pp. 188-195 (October 1994).
11. S.W. Wang and A.E. Kaufman, "Volume Sampled Voxelization of Geometric Primitives", *Visualization '93 Proceedings*, pp. 78-84 (October 1993).
12. T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM* **23**(6) pp. 343-349 (June 1980).
13. J. Wilhelms and A. Van Gelder, "A Coherent Projection Approach for Direct Volume Rendering", Computer Graphics (Proc. SIGGRAPH) **25**(4), pp. 275-284 (July 1991).
14. K.J. Zuiderveld, A.H.J. Koning, and M.A. Viergever, "Acceleration of Ray-Casting Using 3D Distance Transforms", *Visualization and Biomedical Computing Proceedings*, pp. 324-335 (October 1992).

Table 4: Volume Data Set Statistics

Volume Name	In Figure	Size in Samples	Size in Units	Supercell Size	Possibly Contributing Supercells
CT Foot	11	152x261x220	144.4x247.9x220	3x5x5	6.79%
Iron Protein (Positive)	12	66x66x66	66x66x66	1.03x1.03x1.03	5.35%
Iron Protein (Negative)	12	66x66x66	66x66x66	1x1x1	6.2%
Volumetric Floor	13	57x57x12	228x228x48	3.125x3.125x.15	60.93%
Hippocampal Cell	13	384x256x200	306x204.3x160.8	6x4x3.125	3.22%

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE