Efficient Ray Tracing of Volume Data

TR88-029

June 1988

Marc Levoy

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

Efficient Ray Tracing of Volume Data

Marc Levoy

June, 1988

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

Abstract

Volume rendering is a technique for visualizing sampled scalar functions of three spatial dimensions without fitting geometric primitives to the data. Images are generated by computing 2-D projections of a colored semi-transparent volume, where the color and opacity at each point is derived from the data using local operators. Since all voxels participate in the generation of each image, rendering time grows linearly with the size of the dataset. This paper presents a front-to-back image-order volume rendering algorithm and discusses two methods for improving its performance. The first method employs a pyramid of binary volumes to encode coherence present in the data, and the second method uses an opacity threshold to adaptively terminate ray tracing. These methods exhibit a lower complexity than brute-force algorithms, although the actual time saved depends on the data. Examples from two applications are given: medical imaging and molecular graphics.

1. Introduction

The increasing availability of powerful graphics workstations in the scientific and computing communities has catalyzed the development of new methods for visualizing discrete multidimensional data. In this paper, we address the problem of visualizing sampled scalar functions of three spatial dimensions, henceforth referred to as *volume data*. We further focus on a family of visualization methods called *volume rendering techniques* in which a color and an opacity is assigned to each voxel, then a 2-D projection of the resulting colored semi-transparent volume is computed [Levoy88a, Drebin88]. The principal advantages of volume rendering over other visualization methods are its superior image quality and the ability to generate images without explicitly defining surface geometry. The principal drawback of these techniques is their cost. Since all voxels participate in the generation of each image, rendering time grows linearly with the size of the dataset.

This paper presents a front-to-back image-order volume rendering algorithm and discusses two methods for improving its performance. The first optimization is based on the observation that many datasets contain coherent regions of uninteresting voxels. In the context of volume rendering, a voxel is defined as uninteresting if its opacity is zero. Methods for encoding coherence in volume data include octrees [Meagher82], polygonal representations of bounding surfaces [Pizer86], and octree representations of bounding surfaces [Gargantini86]. Methods for taking advantage of coherence during ray tracing include cell decompositions (also known as bounding volumes) [Rubin80] and spatial occupancy enumerations (also known as space subdivisions) [Glassner84]. This paper employs a hierarchical enumeration represented by a pyramid of binary volumes [Yau83]. The pyramid is used to efficiently compute intersections between viewing rays and regions of interest in the data.

The second optimization is based on the observation that once a ray has struck an opaque object or has progressed a sufficient distance through a semi-transparent object, opacity accumulates to a level where the color of the ray stabilizes and ray tracing can be stopped. The idea of adaptively terminating ray tracing was first proposed in [Whitted80]. Many algorithms for displaying medical data stop after encountering the first surface or the first opaque voxel. In this guise, the idea has been reported in [Goldwasser86, Schlusselberg86, Trousset87] and perhaps elsewhere. In volume rendering, surfaces are not explicitly detected. Instead, they appear in the image as a natural byproduct of the stepwise accumulation of color and opacity along each ray. Adaptive termination of ray tracing can thus be implemented by stopping when opacity reaches a user-selected threshold level.

The cost of rendering a volume dataset using a hierarchical enumeration grows linearly with the size of the image, logarithmically with the length of each ray, and linearly with the depth complexity of the data. Adaptive termination of ray tracing reduces the dependence on depth complexity, and for the special case of an environment consisting only of opaque surfaces, eliminates this dependence entirely. In this case, the cost of generating an image grows nearly linearly with the size of the image rather than linearly with the size of the dataset.

2. Brute-force algorithm

Let us first consider the brute-force volume rendering algorithm outlined in figure 1. We begin with a 3-D array of scalar values. For simplicity, let us assume that the array forms a cube measuring N voxels on a side, and that N is an integer power of 2. In this paper, we treat voxels as point samples of a continuous function rather than as volumes of homogeneous value. Voxels are indexed by a vector $\mathbf{i} = (i,j,k)$ where $i,j,k = 1, \ldots, N$, and the value of voxel $\mathbf{i}$ is denoted $f(\mathbf{i})$. Using local operators, a color $C(\mathbf{i})$ and an opacity $\alpha(\mathbf{i})$ is derived for each voxel.

Parallel rays are then traced into the data from an observer position as shown in figure 2. Let us assume that the image is a square measuring P pixels on a side, and that one ray is cast per pixel. Pixels and hence rays are indexed by a vector $\mathbf{u} = (u,v)$ where $u,v = 1, \ldots, P$. For each ray, a vector of colors and opacities is computed by resampling the data at W evenly spaced locations along the ray and tri-linearly interpolating from the colors and opacities in the eight voxels surrounding each sample location. Samples are indexed by a vector $\mathbf{U} = (u,v,w)$ where (u,v) identifies the ray, and $w = 1, \ldots, W$ corresponds to distance along the ray with $w = 1$ being closest to the eye. The color and opacity of sample $\mathbf{U}$ are denoted $C(\mathbf{U})$ and $\alpha(\mathbf{U})$ respectively. Finally, a fully opaque background is draped behind the dataset, and the resampled colors and opacities are composited with each other and with the background to yield a color for the ray. This color is denoted $C(\mathbf{u})$.

The rendering algorithms in [Levoy88a] and [Drebin88] composite in back-to-front order. Specifically, the color $C_{out}(\mathbf{u};\mathbf{U})$ of ray $\mathbf{u}$ as it leaves sample $\mathbf{U}$ on its way toward the eye is related to the color $C_{in}(\mathbf{u};\mathbf{U})$ of the ray as it approaches the sample and the color $C(\mathbf{U})$ and opacity $\alpha(\mathbf{U})$ of the sample by the transparency formula

$$C_{out}(\mathbf{u};\mathbf{U}) = C_{in}(\mathbf{u};\mathbf{U})(1 - \alpha(\mathbf{U})) + C(\mathbf{U})\alpha(\mathbf{U}).$$

where $0 < \alpha < 1$, and $\alpha = 1$ signifies complete attenuation.

In this paper, we composite in front-to-back order. As derived in [Wallace81], the color $C_{post}(\mathbf{u};\mathbf{U})$ and opacity $\alpha_{post}(\mathbf{u};\mathbf{U})$ of ray $\mathbf{u}$ due to the inclusive interval between sample $\mathbf{U}$ and the eye is related to the color $C_{pre}(\mathbf{u};\mathbf{U})$ and opacity $\alpha_{pre}(\mathbf{u};\mathbf{U})$ due to the same interval but excluding the sample by the expression

$$C_{post}(\mathbf{u};\mathbf{U}) = \frac{C_{pre}(\mathbf{u};\mathbf{U})\alpha_{pre}(\mathbf{u};\mathbf{U}) + C(\mathbf{U})\alpha(\mathbf{U})(1 - \alpha_{pre}(\mathbf{u};\mathbf{U}))}{\alpha_{post}(\mathbf{u};\mathbf{U})}$$

where

$$\alpha_{post}(\mathbf{u};\mathbf{U}) = \alpha_{pre}(\mathbf{u};\mathbf{U}) + \alpha(\mathbf{U})(1 - \alpha_{pre}(\mathbf{u};\mathbf{U})).$$

If colors are pre-multiplied by their opacities as suggested in [Porter84], we may replace $C_{post}(\mathbf{u};\mathbf{U})\alpha_{post}(\mathbf{u};\mathbf{U})$ by $\hat{C}_{post}(\mathbf{u};\mathbf{U})$, $C_{pre}(\mathbf{u};\mathbf{U})\alpha_{pre}(\mathbf{u};\mathbf{U})$ by $\hat{C}_{pre}(\mathbf{u};\mathbf{U})$, and $C(\mathbf{U})\alpha(\mathbf{U})$ by $\hat{C}(\mathbf{U})$ in the above expression, yielding the more efficient form

$$\hat{C}_{post}(\mathbf{u};\mathbf{U}) = \hat{C}_{pre}(\mathbf{u};\mathbf{U}) + \hat{C}(\mathbf{U})(1 - \alpha_{pre}(\mathbf{u};\mathbf{U})) \tag{1a}$$

and

$$\alpha_{post}(\mathbf{u};\mathbf{U}) = \alpha_{pre}(\mathbf{u};\mathbf{U}) + \alpha(\mathbf{U})(1 - \alpha_{pre}(\mathbf{u};\mathbf{U})). \tag{1b}$$

After all samples along a ray have been processed, the color $C(\mathbf{u})$ of the ray is obtained from the expression $C(\mathbf{u}) = \hat{C}_{post}(\mathbf{u};\mathbf{W}) / \alpha_{post}(\mathbf{u};\mathbf{W})$ where $\mathbf{W} = (u,v,W)$. If a fully opaque background is draped behind the dataset at $w' = W + 1$, and the ray is composited against this background after passing through the data, then $\alpha_{post}(\mathbf{u};\mathbf{W}') = 1$ where $\mathbf{W}' = (u,v,w')$ and this normalization step can be omitted.

The complete brute-force algorithm is thus summarized as follows.

```
procedure $RayTrace_1(\mathbf{u})$ begin
      $\hat{C}(\mathbf{u}) := 0;$
      $\alpha(\mathbf{u}) := 0;$
      $\mathbf{x}_1 := First(\mathbf{u});$
      $\mathbf{x}_2 := Last(\mathbf{u});$
      $\mathbf{U}_1 := \lceil Image(\mathbf{x}_1) \rceil;$
      $\mathbf{U}_2 := \lfloor Image(\mathbf{x}_2) \rfloor;$
      {Loop through all samples falling within data}
      for $\mathbf{U} := \mathbf{U}_1$ to $\mathbf{U}_2$ do begin
            $\mathbf{x} := Object(\mathbf{U});$
            {Resample color and opacity and composite into ray}
            $\hat{C}(\mathbf{U}) := Sample(\hat{C},\mathbf{x});$
            $\alpha(\mathbf{U}) := Sample(\alpha,\mathbf{x});$
            $\hat{C}(\mathbf{u}) := \hat{C}(\mathbf{u}) + \hat{C}(\mathbf{U})(1 - \alpha(\mathbf{u}));$
            $\alpha(\mathbf{u}) := \alpha(\mathbf{u}) + \alpha(\mathbf{U})(1 - \alpha(\mathbf{u}));$
      end
end $RayTrace_1$.
```

The *First* and *Last* procedures accept a ray index and return the object-space coordinates of the points where the ray enters and leaves the data respectively. These coordinates are denoted by real vectors of the form $\mathbf{x} = (x,y,z)$ where $1 \le x,y,z \le N$. The *Object* and *Image* procedures convert between object-space coordinates and image-space coordinates. The *Sample* procedure accepts a 3-D array of colors or opacities and the object-space coordinates of a point, and returns an approximation to the color or opacity at that point by tri-linearly interpolating from the eight surrounding voxels.

3. Optimized algorithm

The first optimization method we consider is hierarchical spatial occupancy enumeration. We represent this enumeration by a pyramid of M binary volumes as shown in figure 3 where $M = \log_2 N$. Volumes in this pyramid are indexed by a level number m where $m = 0, \ldots, M$, and the volume at level m is denoted V_m. Volume V_0 measures N cells on a side, volume V_1 measures $N/2$ cells on a side, and so on up to volume V_m, which is a single cell. Cells are indexed by a level number m and a vector $\mathbf{i} = (i,j,k)$ where $i,j,k = 1, \ldots, N$, and the value contained in cell $\mathbf{i}$ on level m is denoted $V_m(\mathbf{i})$. We define the size of cells on level m to be 2^m times the spacing between voxels. Since voxels are treated as points, whereas cells fill the space between voxels, each volume is one cell larger in each direction than the underlying dataset as shown in the figure. We also place voxel (1,1,1) at the front-lower-right corner of cell (1,1,1). Thus, for example, cell (1,1,1) on level zero encloses the space between voxels (1,1,1) and (2,2,2).

We construct the pyramid as follows. Cell $\mathbf{i}$ in the base volume V_0 contains a zero if all eight voxels lying at its vertices have opacity equal to zero. Cell $\mathbf{i}$ in any volume V_m, $m > 0$, contains a zero if all eight cells on level $m - 1$ that form its octants contain zeros. In other words, we define

$$V_0(\mathbf{i}) = \begin{cases} 1 & \text{if } \mathbf{i}+\Delta\mathbf{i} \le N \text{ and } \alpha(\mathbf{i}+\Delta\mathbf{i}) = 1 \text{ for any } \Delta\mathbf{i} \text{ where } \Delta\mathbf{i} = (\Delta i,\Delta j,\Delta k) \text{ and } \Delta i,\Delta j,\Delta k = 0,1 \\ 0 & \text{otherwise} \end{cases} \tag{2a}$$

and

$$V_m(\mathbf{i}) = \begin{cases} 1 & \text{if } V_{m-1}(2\mathbf{i}+\Delta\mathbf{i}) = 1 \text{ for any } \Delta\mathbf{i} \text{ where } \Delta\mathbf{i} = (\Delta i,\Delta j,\Delta k) \text{ and } \Delta i,\Delta j,\Delta k = 0,1 \\ 0 & \text{otherwise.} \end{cases} \tag{2b}$$

for $m = 1, \ldots, M$.

We now reformulate our volume rendering algorithm to use this pyramidal data structure. For each ray, we first compute the point where the ray enters the single cell at the top level. We then traverse the pyramid in the following manner. When we enter a cell, we test its value. If it contains a zero, we advance along the ray to the next cell on the same level. If the parent of the new cell differs from the parent of the old cell, we move up to the parent of the new cell. We do this because if the parent of the new cell is unoccupied, we can advance the ray further on our next iteration than if we had remained on a lower level. This ability to advance quickly across uninteresting regions of space is where the algorithm saves its time. If, however, the cell being tested contains a one, we move down one level, entering whichever cell encloses our current location. If we are already at the lowest level, we know that one or more of the eight voxels lying at the vertices of the cell have opacity greater than zero. We then draw samples at evenly spaced locations along that portion of the ray falling within the cell, resample the data at these sample locations, and composite the resulting color and opacity into the color and opacity of the ray.

The second optimization method we consider is adaptive termination of ray tracing. Our goal is to quickly identify the last sample location along a ray that significantly changes the color of the ray. Returning to equation (1a), we define a significant color change as one in which $C_{post}(\mathbf{u};\mathbf{U}) - C_{pre}(\mathbf{u};\mathbf{U}) > \varepsilon$ for some small $\varepsilon > 0$. Since $\alpha_{pre}(\mathbf{u};\mathbf{U})$ increases monotonically along the ray, no significant color changes occur beyond the point where $\alpha_{post}(\mathbf{u};\mathbf{U})$ first exceeds $1 - \varepsilon$. This becomes our termination criterion. Higher values of ε reduce rendering time, while lower values reduce image artifacts. For the datasets used in this paper, $\varepsilon = .05$ usually represents a satisfactory compromise.

Combining both of these optimizations gives us the following algorithm.

```
procedure $RayTrace_2$(u) begin
    $\hat{C}(\mathbf{u}) := 0$;
    $\alpha(\mathbf{u}) := 0$;
    $\mathbf{x} := First(\mathbf{u})$;
    $m := m_{max}$;
    {Loop until beyond data or opacity > threshold}
    while $InBounds(\mathbf{x})$ and $\alpha(\mathbf{u}) \le 1 - \varepsilon$ do begin
        $\mathbf{i} := Index(m,\mathbf{x})$;
        {If high level cell contains a one, drop a level}
        if $V_m(\mathbf{i})$ and $m > m_{\min}$ then $m := m - 1$;
        else begin
            {If level zero cell contains a one, render it}
            if $V_m(\mathbf{i})$ then $Render_2(\mathbf{u},\mathbf{x},Next(m,\mathbf{x}))$;
            {Advance to next cell and maybe jump to higher level}
            while $Parent(m,Index(m,Next(m,\mathbf{x}))) \neq Parent(m,\mathbf{i})$ and $m < M$ do begin
                $\mathbf{i} := Parent(m,\mathbf{i})$;
                $m := m + 1$;
            end
            $\mathbf{x} := Next(m,\mathbf{x})$;
        end
    end
end $RayTrace_2$

procedure $Render_2(\mathbf{u},\mathbf{x}_1,\mathbf{x}_2)$ begin
    $\mathbf{U}_1 := \lceil Image(\mathbf{x}_1) \rceil$;
    $\mathbf{U}_2 := \lfloor Image(\mathbf{x}_2) \rfloor$;
    {Loop through all samples falling within cell}
    for $\mathbf{U} := \mathbf{U}_1$ to $\mathbf{U}_2$ do begin
        $\mathbf{x} := Object(\mathbf{U})$;
        {If any of eight surrounding voxels have opacity > 0,}
        {then resample color and opacity and composite into ray}
        if $V_0(Index(0,\mathbf{x}))$ then begin
            $\hat{C}(\mathbf{U}) := Sample(\hat{C},\mathbf{x})$;
            $\alpha(\mathbf{U}) := Sample(\alpha,\mathbf{x})$;
            $\hat{C}(\mathbf{u}) := \hat{C}(\mathbf{u}) + \hat{C}(\mathbf{U})(1 - \alpha(\mathbf{u}))$;
            $\alpha(\mathbf{u}) := \alpha(\mathbf{u}) + \alpha(\mathbf{U})(1 - \alpha(\mathbf{u}))$;
        end
    end
end $Render_2$.
```

The *Next* procedure accepts a level number and the object-space coordinates of a point along a ray, and returns the coordinates of the point where the ray enters the next cell on the same level. The *Index* procedure accepts a level number and the coordinates of a point, and returns the index of the cell that contains it. The *Parent* procedure accepts a level number and cell index, and returns the index of the parent cell. The algorithm terminates when the ray leaves the pyramid as detected by the *InBounds* procedure.

Figure 4 shows in two dimensions how a typical ray might traverse a hierarchical enumeration. The sequence of points computed by the *Next* procedure are denoted by circular dots in the figure. In regions where the level zero cells contain ones, the spacing between these dots is close to the spacing between voxels. We are therefore led to ask the question: why not simply resample the data at these points? We observe, however, that these points are not evenly spaced along the ray. If the data is resampled at such non-uniformly spaced points, a noise component will be added to the resulting image [Cook86]. To avoid these artifacts, we superimpose a set of evenly spaced sample locations as shown by the rectangular tick marks in the figure, then limit ourselves to resampling the data at these locations.

Assuming that we are rendering a non-empty dataset, most cells on the top levels of the pyramid will contain ones. It is therefore inefficient to begin our traversal there. For the datasets used in this paper, traversal costs were minimized by setting $m_{max} = M - 2$ for all values of M. Assuming an orthographic projection, the cost of advancing a ray from one cell to the next by computing ray-cell intersections is higher than the cost of advancing the ray from one sample location to the next using differencing. It is therefore inefficient to descend to level zero. Instead, we descend to some higher level, loop through the sample locations falling within that cell, and render those for which $V_0(Index(0,\mathbf{x})) = 1$. For the current implementation, $m_{min} = 2$ yields the best results.

4. Discussion

The number of cells in a pyramid of binary volumes is $(8^{M+1}-1)/7$, and each cell requires one bit of computer memory. The cost of accessing a cell is equal to the cost of computing a subscript into a 4-D array. Condensed representations such as octrees or linear octrees [Gargantini82] are also possible, although the amount of memory saved would be small compared to the size of the color and opacity arrays, and the cost of accessing a cell would generally be higher. Since the pyramid depends on the array of opacities rather than on the original data, it must be re-computed whenever these opacities change. The pyramid is independent of observer position, however, and can be used to efficiently generate multiple views from a single set of colors and opacities.

The cost of generating an image using this data structure is the sum of the cost of traversing the pyramid in search of interesting samples, and the cost of resampling the data and compositing at those sample locations. These costs are highly dependent on the data and the viewing angle. In this paper, we will consider datasets consisting of a number of opaque or semi-transparent surfaces. Using methods described in [Levoy88a], a plot of opacity along a line perpendicular to one of these surfaces typically exhibits a bump shape of some finite width T.

Let us analyze the cost of rendering a volume containing S such surfaces, each parallel to each other, parallel to the plane defined by two of the three coordinate axes, and spaced N/S voxels apart as shown in figure 5. In this case, we may represent the pyramid by a binary tree containing N leaf nodes and horizontal connections on every level. Let us assume that our viewing rays are perpendicular to the surfaces and that the spacing between samples along a ray is equal to the spacing between voxels in object space. The cost of finding the first interesting sample on surface 1 using the algorithm described in section 3 is proportional to the length of path A through the tree. This length is clearly $O(\log_2 N)$. The cost of resampling and compositing surface 1 is proportional to its width T. The cost of advancing from the last interesting sample on surface 1 to the first interesting sample on surface 2 is proportional to the length of path B, which is also $O(\log_2 N)$. The total cost of tracing the ray is thus $O(S\log_2 N + S)$, which is logarithmic in the length of the ray and linear in the depth complexity of the data.

Adaptive termination of ray tracing reduces the dependence on depth complexity by some constant that depends on the data. For the special case of an environment consisting only of opaque surfaces, rendering cost is proportional to the cost of finding, resampling, and compositing surface 1, which is logarithmic in the length of the ray but independent of depth complexity. The cost of generating an image therefore grows nearly linearly with the size of the image rather than linearly with the size of the dataset.

5. Implementation and results

To understand how the algorithm behaves on real data, let us consider some examples. The characteristics of three datasets are given in table 1. The first is a computed tomography (CT) study of a human skull mounted in a lucite head cast. To demonstrate the effect of semi-transparent surfaces on the performance of the algorithm, this dataset was rendered twice, once with a semi-transparent air-lucite boundary surface (figure 6), and once with a completely transparent boundary surface (figure 7). The second dataset is a portion of an electron density map of Staphylococcus Aureus ribonuclease. A volume rendering of an isovalue surface from this map is shown in figure 8. The polymer backbone crosses the image from bottom to top, and two Tyrosine residues with their characteristic six-atom benzene rings can be seen extending to the left and right sides of the backbone. A color-coded stick representation of the molecular structure has been superimposed on the image to aid in its interpretation. To study the growth of rendering cost with respect to dataset size, this dataset was rendered at three different spatial resolutions, the largest of which is shown in the figure. The last dataset is a CT study of a complete human head, a volume rendering of which is shown in figure 9.

Rendering times for these datasets are summarized in table 2. Separate entries are provided for the brute-force algorithm, the optimized algorithm with adaptive termination of ray tracing disabled by setting $\varepsilon = 0$, and the fully optimized algorithm with $\varepsilon = .05$. All algorithms were implemented in the C language on a Sun 4/280 with 32MB of main memory. As the table shows, hierarchical enumeration reduced rendering time by a factor of between 2.0 and 5.0 for this data, and adaptive termination of ray tracing added another factor of between 1.3 and 2.2. We also observe that adding a semi-transparent surface to the rendering of the skull fragment decreased the amount of time saved but did not eliminate the savings completely. We finally note that doubling the width of the electron density map increased rendering time by roughly a factor of eight for the brute-force algorithm and five for the optimized algorithm. These ratios are in close agreement with the analysis suggested in section 4.

To help us interpret these results, the cost of generating figure 9 has been broken down into its constituent parts. Using the brute-force rendering algorithm described in section 2, the cost of finding all interesting samples along a ray is proportional to the length of the ray clipped to the boundaries of the dataset. For the observer position used in figure 9, a visualization of this cost is shown in figure 10a. Brighter pixels represent more work. The image is essentially an X-ray of a cube of uniform density. The cost of resampling and compositing the interesting samples along a ray is proportional to the number found along the ray. For the dataset under consideration, a visualization of this cost is shown in figure 10b. This image is essentially an X-ray of a binary representation of the data. As expected, it is brightest along silhouettes where rays pass through large amounts of bony material. The total cost of rendering figure 9 using the brute-force algorithm is a weighted sum of figures 10a and 10b.

Using hierarchical enumeration, the cost of finding all interesting samples along a ray is proportional to the number of iterations through the outer loop in the *RayTrace*$_2$ procedure plus the number of tests of level zero cells performed in the *Render*$_2$ procedure. A visualization of this cost is shown in figure 11a. This image is essentially an X-ray of an octree. The cost of resampling and compositing the interesting samples is shown in figure 11b. Since the use of enumeration alone does not reduce the number of samples composited, figure 11b is identical to figure 10b. The total cost of rendering figure 9 using a hierarchical enumeration is a weighted sum of figures 11a and 11b.

Adaptive termination of ray tracing reduces the number of interesting samples which must be found. For $\varepsilon = .05$, a visualization of the reduced cost is shown in figure 12a. In regions where fewer samples are processed, resampling and compositing costs drop as well, as shown in figure 12b. The total cost of rendering figure 9 using both of the optimization methods is a weighted sum of figures 12a and 12b.

6. Conclusions

A method for efficiently visualizing sampled scalar functions of three spatial dimensions has been described. The method employs both hierarchical spatial occupancy enumeration and adaptive termination of ray tracing to reduce the complexity of the rendering problem. Although the amount of time saved depends on the data and the viewing angle, savings of more than an order of magnitude have been observed for many datasets.

If there is coherence present in a dataset, there may also be coherence present in its projections. This is particularly true for data acquired from sensing devices where the original scene is bandlimited prior to digitization. We can take advantage of this bandlimiting by varying the number of rays cast as a function of local image complexity [Levoy88b]. In many cases, this optimization reduces rendering time by another order of magnitude.

Any combination of data and opacity assignment operators that partitions a volume dataset into coherent regions of opaque and transparent voxels is a candidate for the methods presented in this paper. Although we have only explored the visualization of surfaces, these optimization techniques are very general and should find use in many scientific and diagnostic applications.

7. Acknowledgements

The author wishes to thank Profs. Henry Fuchs, Steven M. Pizer, Frederick P. Brooks Jr., and Turner Whitted of the Computer Science Department and Drs. Julian Rosenman and Edward L. Chaney of the Radiation Oncology Department for their encouragement and support. Thanks are also due to John Gauch for many enlightening discussions. The CT scans used in this paper were provided by the Radiation Oncology Department at North Carolina Memorial Hospital. The electron density map was provided by Dr. Chris Hill of the University of York Chemistry Department, and was reformatted and brought on-line with the help of Mark Harris of the University of North Carolina. This work was supported by ONR grant N00014-86-K-0680 and NIH grant R01-CA39060.

8. References

[Cook86] Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics*, Vol. 5, No. 1, January, 1986, pp. 51-72.

[Drebin88] Drebin, R.A., Carpenter, L., Hanrahan, P., "Volume Rendering," *Computer Graphics* (to appear).

[Gargantini82] Gargantini, I., "Linear Octtrees for Fast Processing of Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 20, 1982, pp. 365-374.

[Gargantini86] Gargantini, I., Walsh, T.R.S., and Wu, O.L., "Displaying a Voxel-Based Object via Linear Octtrees," *Proceedings SPIE*, Vol. 626, 1986, pp. 460-466.

[Glassner84] Glassner, A.S., "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, Vol. 4, No. 10, October, 1984, pp. 15-22.

[Goldwasser86] Goldwasser, Samuel, "Rapid Techniques for the Display and Manipulation of 3-D Biomedical Data," *Tutorial presented at 7th Annual Conference of the NCGA*, Anaheim, CA, May, 1986.

[Levoy88a] Levoy, M., "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May, 1988, pp. 29-37.

[Levoy88b] Levoy, M., "Volume Rendering by Adaptive Refinement," Technical Report 88-030, Computer Science Department, University of North Carolina at Chapel Hill, June, 1988.

[Meagher82] Meagher, D., "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, Vol. 19, 1982, pp. 129-147.

[Pizer86] Pizer, S.M., Fuchs, H., Mosher, C., Lifshitz, L., Abram, G.D., Ramanathan, S., Whitney, B.T., Rosenman, J.G., Staab, E.V., Chaney, E.L. and Sherouse, G., "3-D Shaded Graphics in Radiotherapy and Diagnostic Imaging," *NCGA '86 conference proceedings*, Anaheim, CA, May, 1986, pp. 107-113.

[Porter84] Porter, Thomas and Duff, Tom, "Compositing Digital Images," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 253-259.

[Rubin80] Rubin, Steven M. and Whitted, Turner, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics*, Vol. 14, No. 3, July 1980, pp. 110-116.

[Schlusselberg86] Schlusselberg, Daniel S. and Smith, Wade K., "Three-Dimensional Display of Medical Image Volumes," *Proceedings of the 7th Annual Conference of the NCGA*, Anaheim, CA, May, 1986, Vol. III, pp. 114-123.

[Trousset87] Trousset, Yves and Schmitt, Francis, "Active-Ray Tracing for 3D Medical Imaging," *EUROGRAPHICS '87 conference proceedings*, pp. 139-149.

[Wallace81] Wallace, B.A., "Merging and Transformation of Raster Images for Cartoon Animation," *Computer Graphics*, Vol. 15, No. 3, August, 1981, pp. 253-262.

[Whitted80] Whitted, T., "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, Vol. 23., No. 6, June, 1980, pp. 343-349.

[Yau83] Yau, M., and Srihari, S.N., "A Hierarchical Data Structure for Multidimensional Digital Images," *Communications of the ACM*, Vol. 26., No. 7, July, 1983, pp. 504-515.

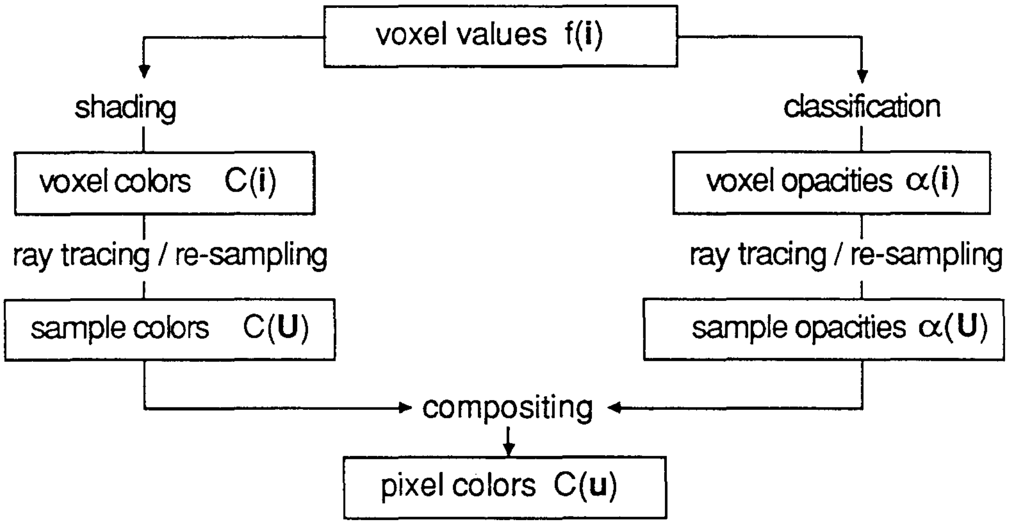

Figure 1 - Overview of volume rendering algorithm

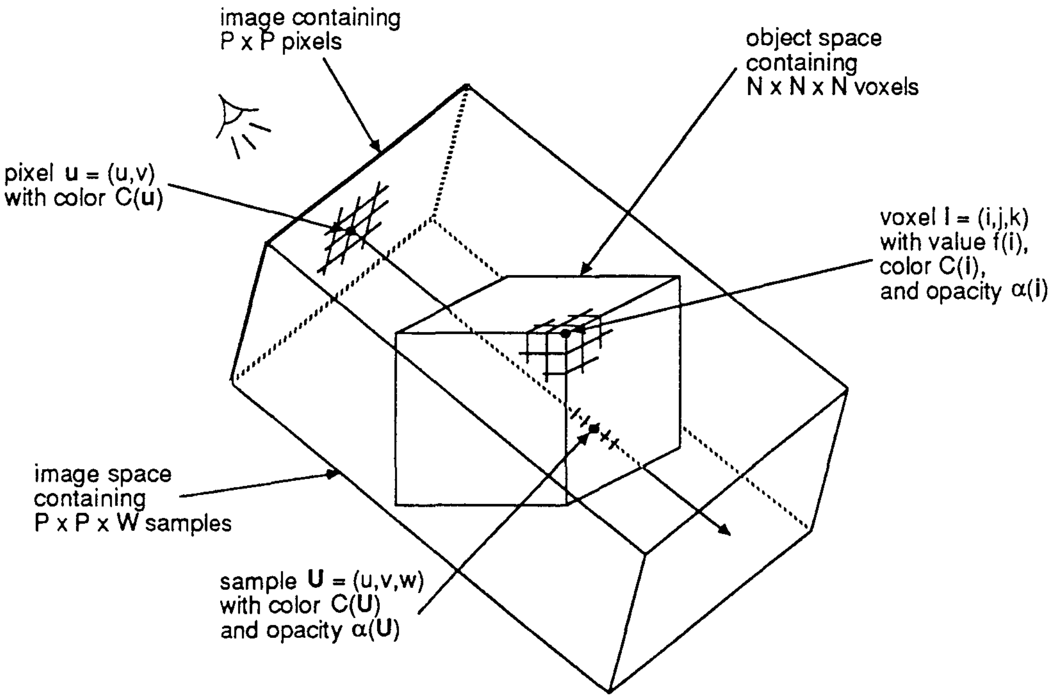
image containing P x P pixels
object space containing N x N x N voxels
pixel u = (u,v) with color C(u)
voxel I = (i,j,k) with value f(i), color C(i), and opacity α(i)
image space containing P x P x W samples
sample U = (u,v,w) with color C(U) and opacity α(U)

Figure 2 - Relationship between object space and image space

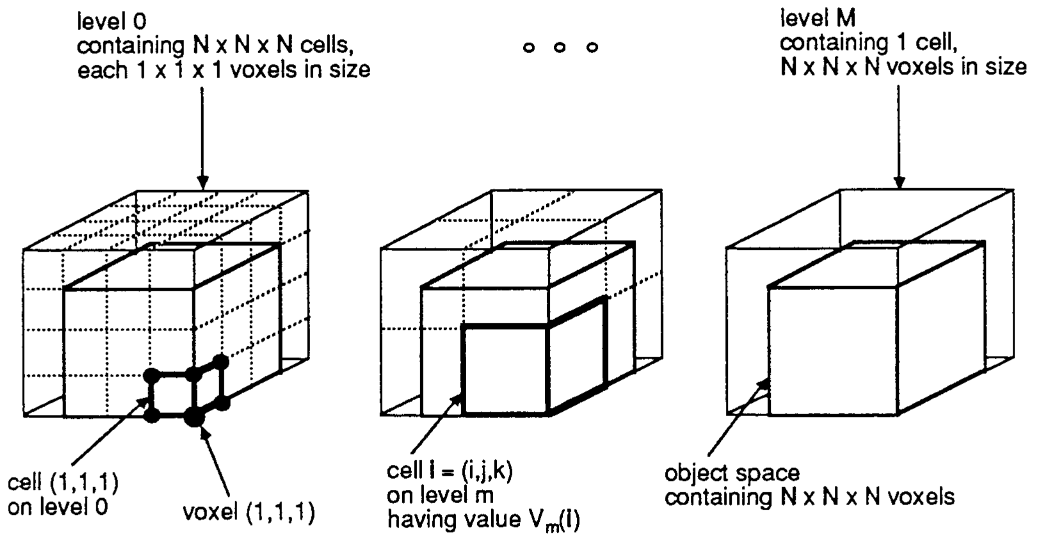

Figure 3 - Hierarchical enumeration of object space

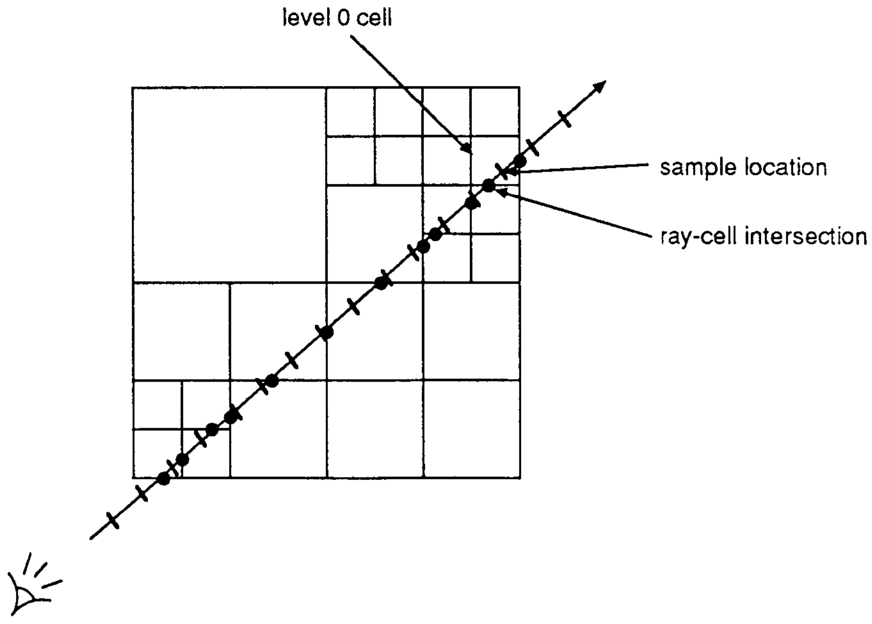

Figure 4 - Ray tracing of hierarchical enumeration

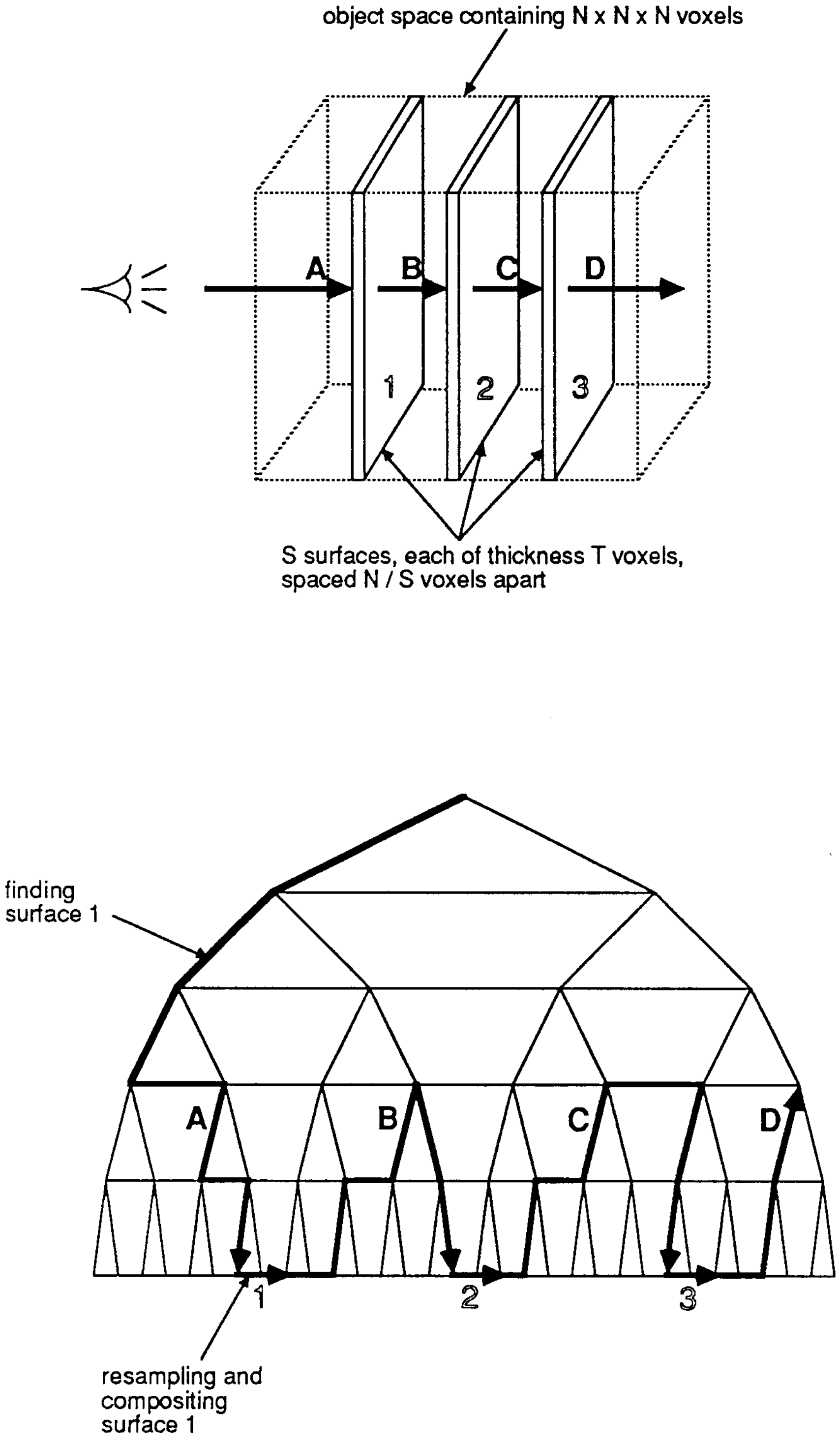
object space containing N x N x N voxels
A
B
C
D
1
2
3
S surfaces, each of thickness T voxels, spaced N / S voxels apart
finding surface 1
A
B
C
D
1
2
3
resampling and compositing surface 1

Figure 5 - Cost of traversing a hierarchical enumeration