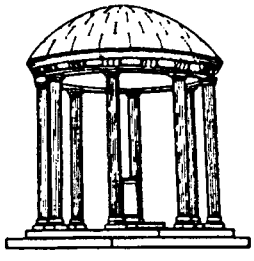
Volume Rendering by Adaptive Refinement

TR88-030

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



Volume Rendering by Adaptive Refinement

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. This paper presents an image-order volume rendering algorithm in which image quality is adaptively refined over time. The first image is generated by casting a small number of rays into the data. The usefulness of these rays is maximized by distributing them according to measures of local image complexity. An image is formed from the resulting non-uniform distribution of colors by interpolation. Subsequent images are generated by alternately casting more rays and interpolating. Examples from two applications are given: molecular graphics and medical imaging.

1. Introduction

In this paper, we address the problem of interactively visualizing sampled scalar functions of three spatial dimensions, henceforth referred to as *volume data.* We focus on a rapidly growing 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.

The complexity of volume rendering can be reduced by taking advantage of object-space coherence. Methods based on hierarchical spatial enumeration and adaptive termination of ray tracing are presented in [Levoy88b]. 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. Nevertheless, the time required to render a large dataset using currently available workstation technology is on the order of tens of seconds or minutes. To be useful in an interactive setting, reductions of at least another order of magnitude are desirable.

This paper presents an image-order volume rendering algorithm in which image quality is adaptively refined over time. A survey of refinement techniques for polygonal environments is given in [Bergman86]. These techniques have three principal characteristics: they distribute work according to where it makes the most difference, they form intermediate images from partial information, and they minimize the amount of work discarded after formation of these images. In the present paper, an initial level of detail is selected and a number of rays are cast. The usefulness of

these rays is maximized by distributing them according to measures of local image complexity. In classical ray tracing, methods for distributing rays non-uniformly include recursive subdivision of image space [Whitted80] and stochastic sampling [Lee85, Dippe85, Cook86, Kajiya86]. Methods for measuring local image complexity include color differences [Whitted80] and statistical variance [Lee85, Kajiya86]. This paper employs recursive subdivision based on local color differences.

Typical ray densities for polygonal environments range between one and one hundred rays per pixel. Images are formed by adding the contributions from all rays cast in each pixel. Volume data, however, is assumed bandlimited before sampling. Additional blurring is often introduced during acquisition, as in the case of computed tomography (CT) and electron density maps obtained from X-ray diffraction data [Herman80, Glusker85]. Certain amorphous phenomena occurring in astronomy and physics are even more bandlimited [Upson86]. As a result, the spacing between pixels is typically set equal to the spacing between voxels, and no more than one ray is cast per pixel. This leaves us with the problem of forming intermediate images from a non-uniform and possibly sparse array of colors. In this paper, images are formed by recursively bi-linearly interpolating between available colors and resampling at the display resolution.

Following image formation, interpolated colors are discarded, the level of detail is increased, and more rays are cast. Since rays cast in previous steps are not discarded, the cost of computing subsequent images is equal to the cost of casting additional rays added to the cost of repeating the recursive subdivision and interpolation operations. In the current implementation, these latter costs are small compared to the cost of ray tracing. The amount of work discarded after image formation is therefore minimal.

Using this method, crude images of many datasets can be obtained in few seconds. Gradually better images are obtained at intervals of a few seconds each, culminating in a high quality image in less than a minute.

2. Description of algorithm

The volume rendering algorithm used in this paper is summarized 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. 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 as described in [Levoy88a].

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 $u,v = 1, \ldots, R$. We divide the image into square regions measuring ω_{max} pixels on a side where $1 \le \omega_{max} \le R$, then cast rays from the four corner pixels of each region. 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. 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 $C(\mathbf{u})$.

If the range of colors returned by the four rays in a region is less than some ε where $\varepsilon > 0$, no further processing is performed on this region. Otherwise, the region is divided into four subregions and more rays are cast. Subdivision continues until the range of colors falls below ε or the size of the region reaches some ω_{min} where $1 \le \omega_{min} \le \omega_{max}$. An array of flags $F(\mathbf{u})$ is also maintained, and rays are traced only from pixels whose flags are clear. Once a ray has been traced, its

flag is set. This avoids duplication of work when subsequent regions are processed, and also when the current region is processed again to compute the next image.

When ray tracing is complete, an image is formed by bi-linearly interpolating between available colors and resampling. To insure continuity despite the non-uniform distribution of rays, a recursive method similar to the algorithm employed during ray tracing is used. The image is divided into square regions measuring ω_{max} pixels on a side. Pixels are interpolated at the midpoints of the four sides and at the center of each region. The region is then divided into four subregions and the process is repeated. Subdivision continues until the region contains a single pixel. An array $G(\mathbf{u})$ of flags is maintained, and interpolation is performed only at pixels that have neither been ray traced nor interpolated, i.e. pixels whose F and G flags are clear. Once a pixel has been interpolated, its G flag is set.

When interpolation is complete, the image is displayed. To continue the refinement process, the image is cleared of all interpolated colors, i.e. pixels whose F flag is clear, the level of detail is raised by decreasing ω_{min}, ω_{max}, or ε, and ray tracing is begun anew. The process terminates when $\omega_{max} = 1$, or when the user stops it. The complete algorithm is summarized as follows:

```
procedure Refine() begin
    for v := 1 to R do for u := 1 to R do begin
            C(u,v) := 0;  α(u,v) := 0;  F(u,v) := 0;  G(u,v) := 0;
    end
    {Initialize level-of-detail parameters}
    ωmax := Firstωmax();  ωmin := Firstωmin();  ε := Firstε();
    {Loop until image is fully refined}
    while ωmax > 1 do begin
            {Cast some (more) rays, then form an image by interpolation}
            RayTraceData(ωmax,ωmin,ε);
            InterpolateImage(ωmax);
            DisplayImage();
            {Clear all interpolated colors}
            for v := 1 to R do for u := 1 to R do begin
                    if not F(u,v) then C(u,v) := 0;  G(u,v) := 0;
            end
            {Increment level-of-detail parameters}
            ωmax := Nextωmax();  ωmin := Nextωmin();  ε := Nextε();
    end
end Refine

procedure RayTraceData(ωmax,ωmin,ε) begin
    {Divide image into square regions for ray tracing}
    for v := 1 to R by ωmax do for u := 1 to R by ωmax do
            RayTraceRegion(u,v,ωmax,ωmin,ε);
end RayTraceData
```

```
procedure RayTraceRegion(u,v,ω,ω_min,ε) begin
    {Cast rays from four corners of region}
    for j := 0 to ω by ω do for i := 0 to ω by ω do
        if u+i ≤ R and v+j ≤ R and not F(u+i,v+j) then begin
            RayTrace(u+i,v+j);  F(u+i,v+j) := 1;
        end
    {If color difference > threshold and region is larger than ω_min by ω_min pixels,}
    {divide into four subregions and continue ray tracing}
    if ω > ω_min and Difference(u,v,ω,ε) then
        for j := 0 to ω/2 by ω/2 do for i := 0 to ω/2 by ω/2 do
            RayTraceRegion(u+i,v+j,ω/2,ω_min,epsilon);
end RayTraceRegion

procedure InterpolateImage(ω_max) begin
    {Divide image into square regions for interpolation}
    for v := 1 to R by ω_max do for u := 1 to R by ω_max do
        InterpolateRegion(u,v,ω_max);
end InterpolateImage

procedure InterpolateRegion(u,v,ω) begin
    {Interpolate colors at midpoints of sides and at center of region}
    for j := 0 to ω by ω/2 do for i := 0 to ω by ω/2 do
        if u+i ≤ R and v+j ≤ R and not F(u+i,v+j) and not G(u+i,v+j) then begin
            Interpolate(u,v,ω,u+i,v+j);  G(u+i,v+j) := 1;
        end
    {If region is larger than 2 x 2 pixels,}
    {divide into four subregions and continue interpolation}
    if ω/2 > 1 then
        for j := 0 to ω/2 by ω/2 do for i := 0 to ω/2 by ω/2 do
            InterpolateRegion(u+i,v+j,ω/2);
end InterpolateRegion.
```

The *First* and *Next* procedures respectively initialize and increment the level-of-detail parameters according to some user-selected sequence. The *RayTrace* procedure traces a ray from pixel $(u+i,v+j)$ into the data and loads the resulting color $C(u+i,v+j)$ into the image array. The *Difference* procedure decides if the range of intensity values for any component (red, green, or blue) of the colors at the four corners of the specified region exceed ε. The *Interpolate* procedure computes a color $C(u+i,v+j)$ for pixel $(u+i,v+j)$ using linear interpolation (in the case of region boundary midpoints) or bi-linear interpolation (in the case of region centers) and loads it into the image array. The *DisplayImage* procedure displays the interpolated image.

3. Implementation and results

To demonstrate the performance of this algorithm, two case studies are presented. The first is a 123 x 123 x 123 voxel portion of an electron density map of cytochrome B5. Using methods described in [Levoy88a], an isovalue surface was selected for display. Using the algorithm described in section 2 of this paper, a four-frame adaptive refinement sequence was then generated. The resulting images are shown in figure 3, with the sequence running from top-left to bottom-right. While the upper-left image exhibits obvious artifacts, it is still useful. The other three images appear at intervals of approximately five seconds each using an implementation in the C language on a Sun 4/280 with 32MB of main memory.

Performance statistics for this case study are summarized in table 1. For each image, the table gives values for the three level-of-detail parameters, incremental and total ray counts, and incremental and total elapsed times. Almost all of the elapsed time is due to ray tracing. Subdivision and interpolation take fractions of a second each.

A visualization of the F array showing where rays were cast is given in figure 4. Each white pixel in this figure corresponds to a single ray. Thus, any pixel in figure 3 whose corresponding pixel in figure 4 is white was computed by ray tracing, whereas any pixel in figure 3 whose figure 4 pixel is black was computed by interpolation. As expected, ray densities are highest along surface silhouettes, where color differences are highest. Since $\omega_{max} = 1$ in the last frame of the sequence (lower-right image), rays are cast from every pixel. The F array for this case is completely white.

The second dataset is a CT scan of a human head and was acquired as 113 slices of 256 x 256 samples each. Using methods described in [Levoy88a], the bone surface was selected for display. A four-frame adaptive refinement sequence was then generated, the first frame of which is shown in figure 5 and the last frame of which is shown in figure 6. Performance statistics for this dataset are summarized in table 2.

4. Conclusions

A method for interactively visualizing sampled scalar functions of three spatial dimensions has been described. Although ray tracing is not generally considered a candidate rendering algorithm for interactive systems, it is used here because of the superiority of the images it produces. Assuming that rays can be traced efficiently, the incremental nature of ray casting lends itself well to an interactive system based on adaptive refinement.

In this study, values for the three level-of-detail parameters ω_{max}, ω_{min}, and ε were selected manually. Generally speaking, high values of ω_{max} cause features to be missed, high values of ε cause features to be ignored even if they are not missed, and high values of ω_{min} causes features to be poorly resolved even if they are neither missed nor ignored. Inappropriate values for these parameters cause sub-optimal presentation of the data as well as unequal intervals between successive frames in refinement sequences. Algorithms are needed that automatically select an optimum sequence of values based on the characteristics of a particular dataset.

5. 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 electron density map used in this study was obtained from Jane and Dave Richardson of Duke University, and the CT scan was

provided by the Radiation Oncology Department at North Carolina Memorial Hospital. This work was supported by ONR grant N00014-86-K-0680 and NIH grant R01-CA39060.

6. References

[Bergman86] Bergman, L., Fuchs, H., Grant, E., Spach, S., "Image Rendering by Adaptive Refinement," *Computer Graphics,* Vol. 20, No. 4, August, 1986, pp. 29-37.

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

[Dippe85] Dippe, M.A.Z., Wold, E.H., "Antialiasing Through Stoachastic Sampling," *Computer Graphics,* Vol. 19, No. 3, July, 1985, pp. 69-78.

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

[Glusker85] Glusker, P.J., Trueblood, K.N., *Crystal Structure Analysis,* Oxford University Press, Oxford, 1985.

[Herman80] Herman, G.T., *Image Reconstruction from Projections,* Academic Press, New York, 1980.

[Kajiya86] Kajiya, J.T., "The Rendering Equation," *Computer Graphics,* Vol. 20, No. 4, August, 1986, pp. 143-150.

[Lee85] Lee, M.E., Redner, R.A., Uselton, S.P., "Statistically Optimized Samling for Distributed Ray Tracing," *Computer Graphics,* Vol. 19, No. 3, July, 1985, pp. 61-67.

[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., "Efficient Ray Tracing of Volume Data," Technical Report 88-029, Computer Science Department, University of North Carolina at Chapel Hill, June, 1988.

[Upson86] Upson, C., "The Visual Simulation of Amorphous Phenomena," *The Visual Computer,* Vol. 2, 1986, pp. 321-326.

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

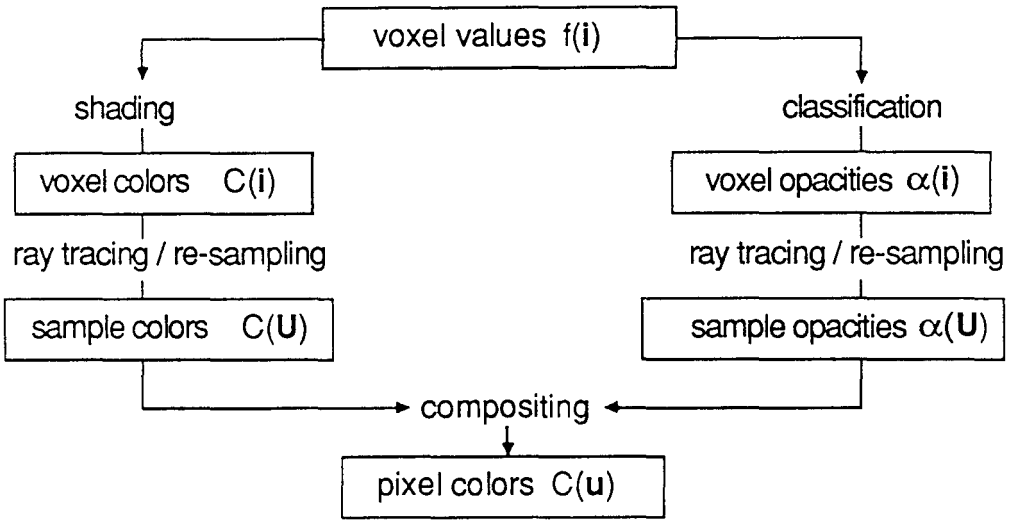


Figure 1 - Overview of volume rendering algorithm

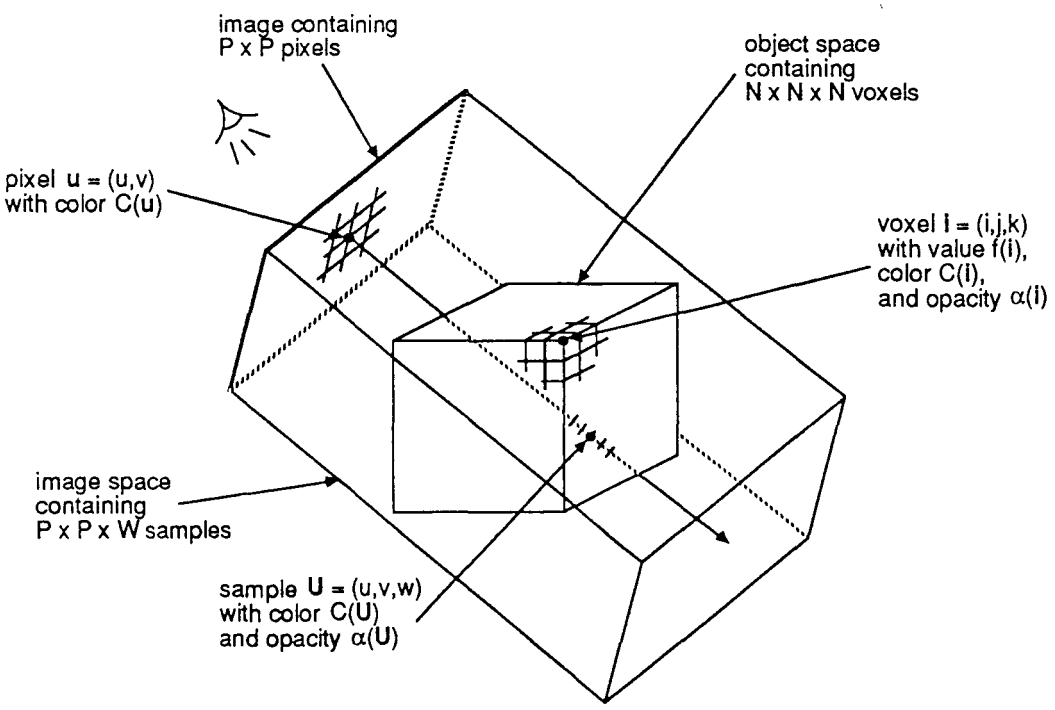


Figure 2 - Relationship between object space and image space

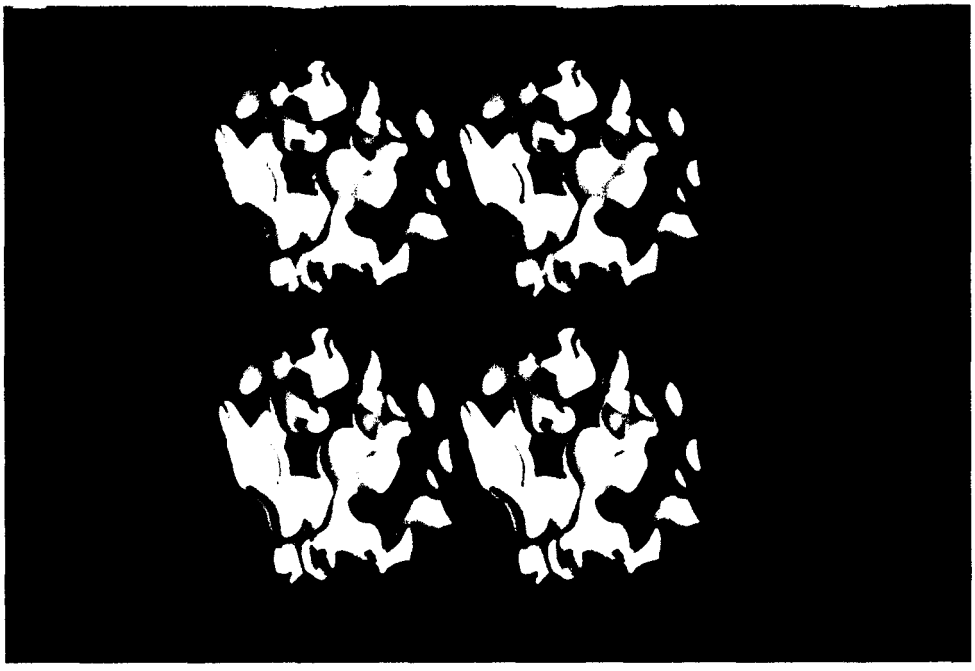


Figure 3 - Adaptively refined volume rendering of cytochrome

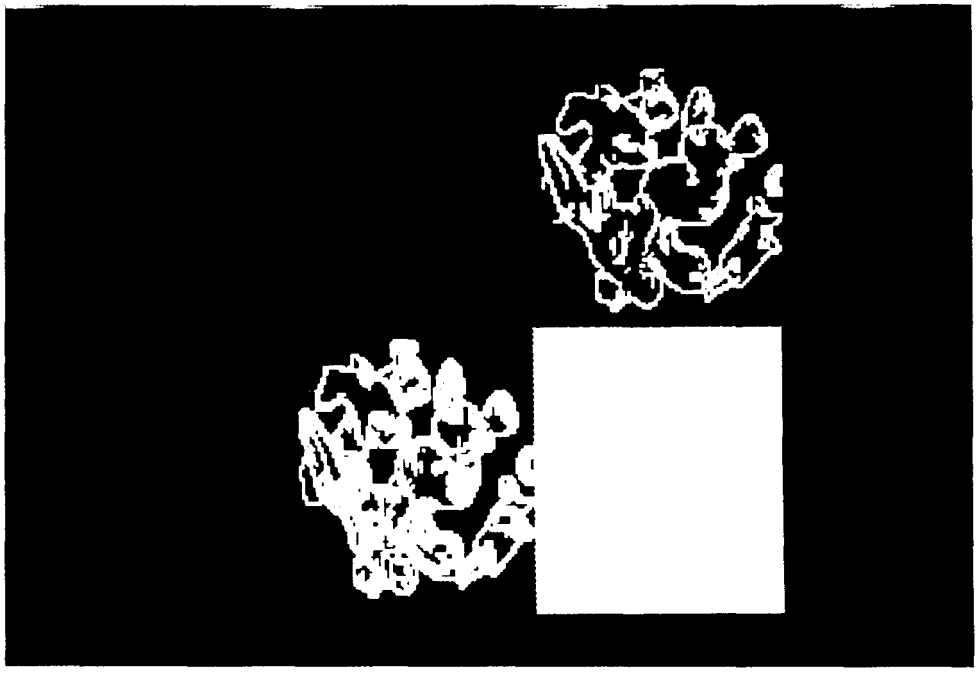


Figure 4 - Visualization of where rays were cast to generate figure 3

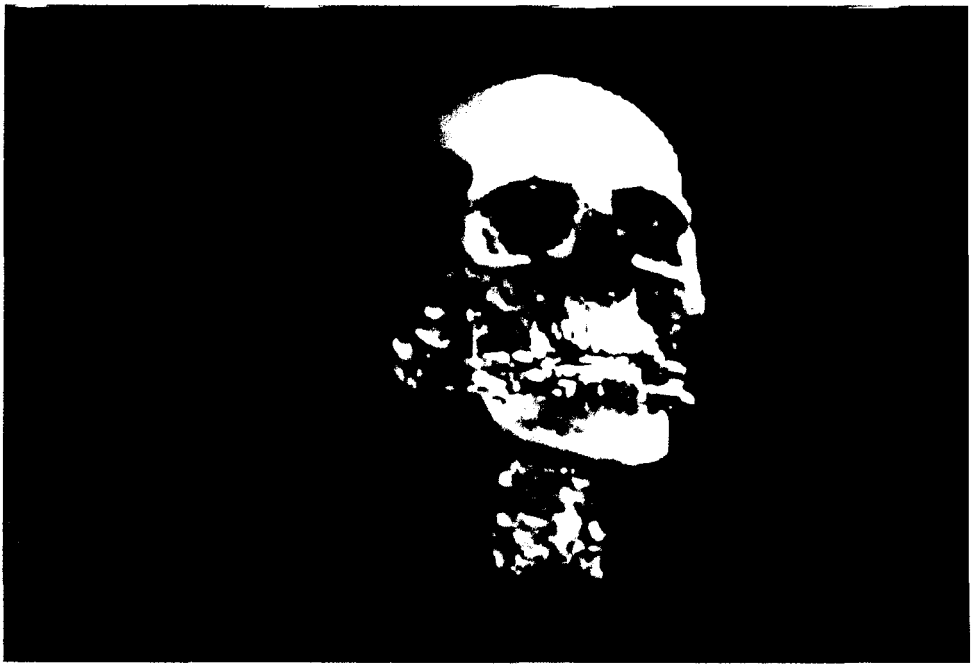


Figure 5 - Adaptively refined volume rendering of head - first frame

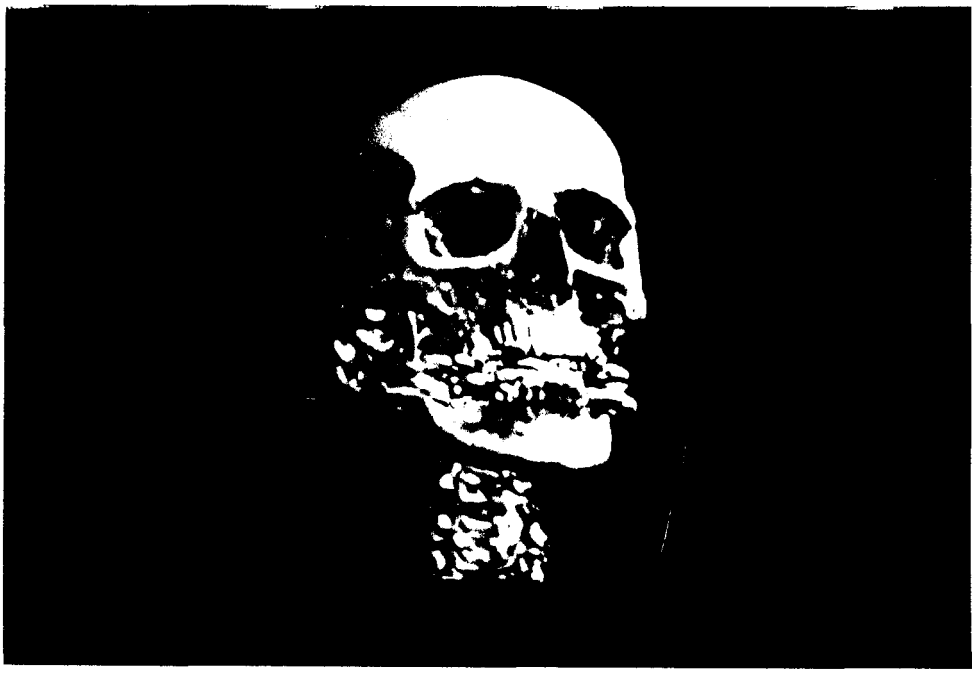


Figure 6 - Adaptively refined volume rendering of head - fourth frame

frame	image fig	rays fig	total rays	additional rays	%	total time	time interval	ω_{max}	ω_{min}	ε
1	3a	4a	3,870	3,870	20	4.6 secs	4.6 secs	16	2	32
2	3b	4b	9,947	6,077	51	12.1	7.5	8	1	32
3	3c	4c	14,459	4,512	74	17.1	5.0	4	1	16
4	3d	4d	19,289	4,830	100	25.7	8.6	1	1	-

Table 1 - Performance statistics for cytochrome

frame	image fig	rays fig	total rays	additional rays	%	total time	time interval	ω_{max}	ω_{min}	ε
1	5	-	5,179	5,179	16	13 secs	13 secs	16	2	16
2	-	-	16,136	10,957	52	41	28	8	1	16
3	-	-	21,625	5,489	70	63	22	2	1	12
4	6	-	30,603	8,978	100	104	41	1	1	-

Table 2 - Performance statistics for head

Figure 6 - Volume rendering of jaw with semi-tranparent skin

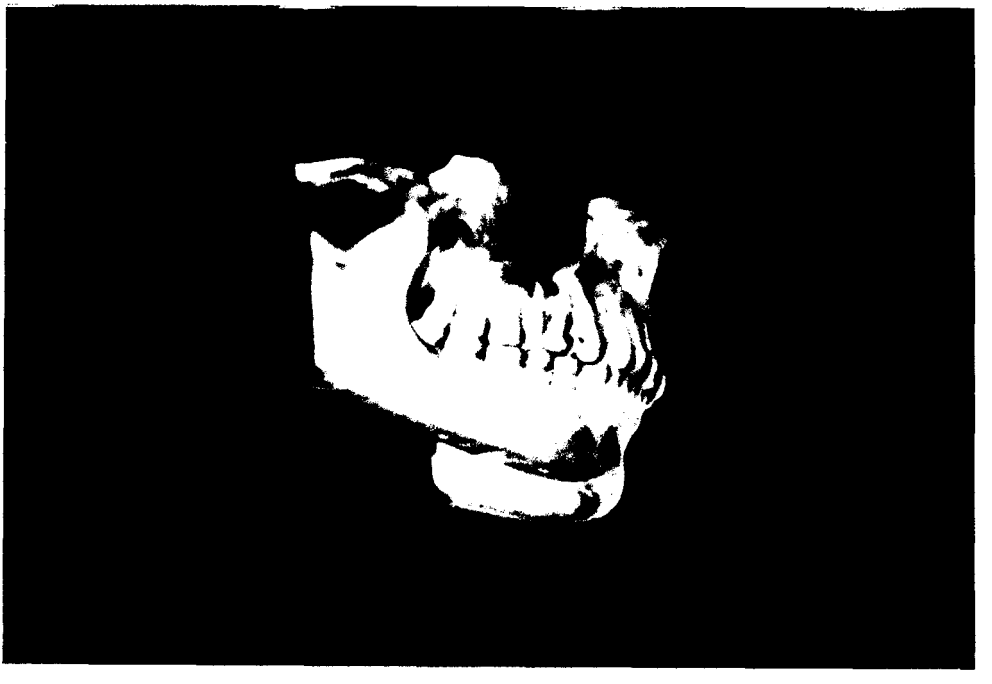
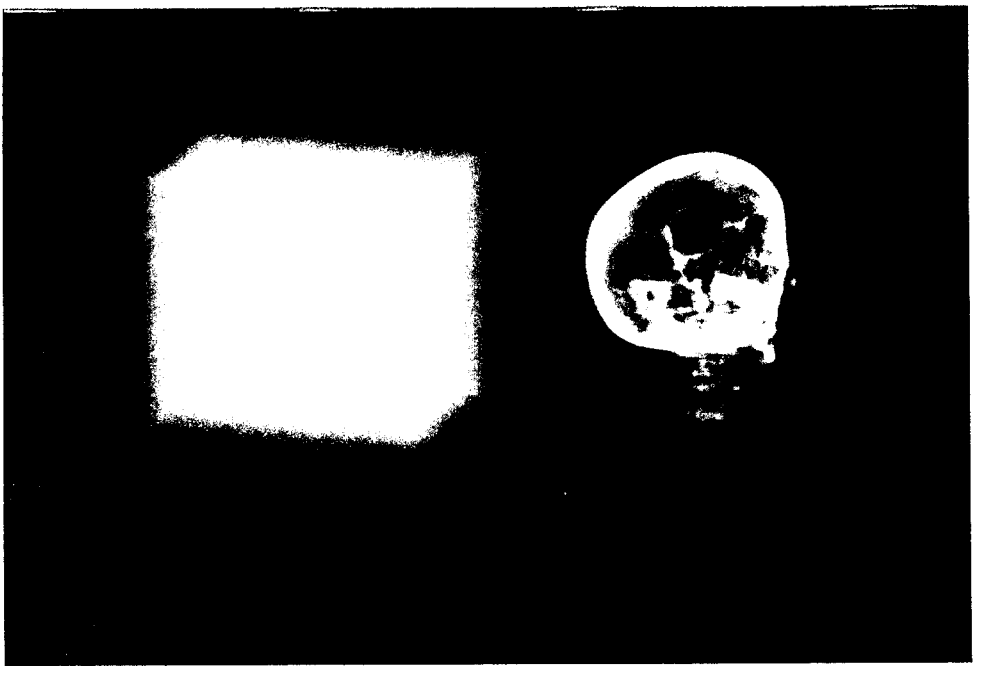


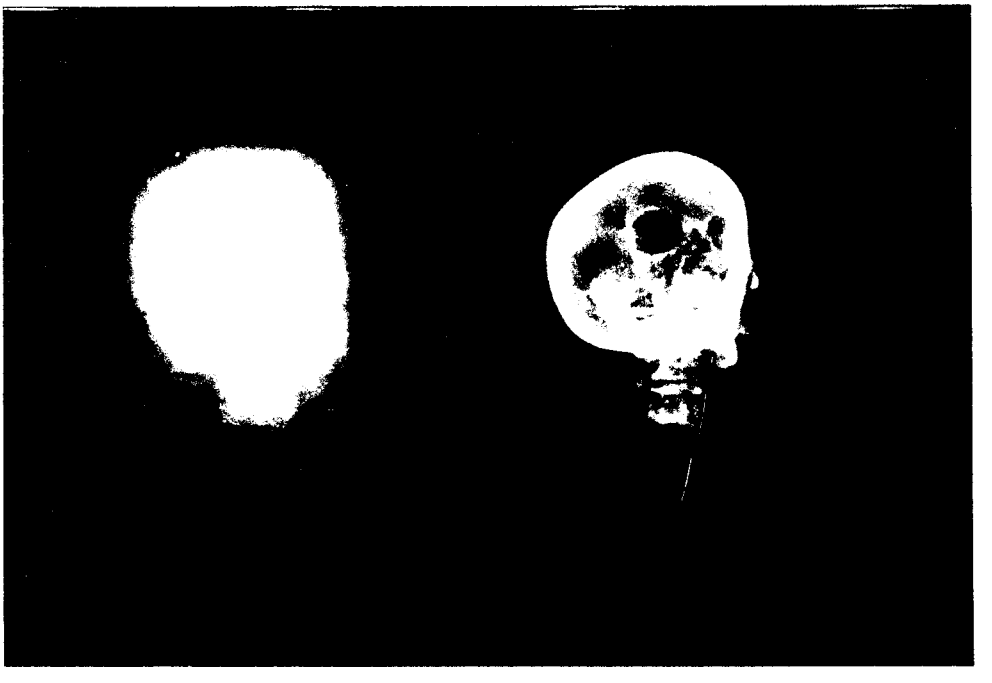
Figure 7 - Volume rendering of jaw without skin

Figure 8 - Volume rendering of ribonuclease

Figure 9 - Volume rendering of head



Figures 10a and 10b - Costs of rendering figure 9
using brute-force algorithm



Figures 11a and 11b - Costs of rendering figure 9
using hierarchical enumeration

Figures 12a and 12b - Costs of rendering figure 9
using hierarchical enumeration and adaptive termination of ray tracing

name	fig	acquired size	scaling factor	size after scaling	samples drawn	samples with $\alpha > 0$	%
jaw with skin	6	256 x 128 x 59	1 x 1 x 2	256 x 128 x 113	3,541,851	594,472	17
jaw w/o skin	7	256 x 128 x 59	1 x 1 x 2	256 x 128 x 113	3,541,851	335,751	9
ribonuclease	8	24 x 20 x 11	12 x 12 x 12	288 x 244 x 132	7,067,842	810,542	11
ribonuclease	-	24 x 20 x 11	6 x 6 x 6	144 x 120 x 66	825,465	160,747	19
ribonuclease	-	24 x 20 x 11	3 x 3 x 3	72 x 60 x 33	92,724	25,531	27
head	9	256 x 256 x 113	1 x 1 x 2	256 x 256 x 226	14,081,917	1,249,458	9

Table 1 - Characteristics of datasets

name	fig	brute-force	hierarchical enumeration	enumeration and adaptive termination	col. 1 / col. 2	col. 2 / col. 3	col. 1 / col. 3
jaw with skin	6	293 secs	94 secs	57 secs	3.1	1.6	5.1
jaw w/o skin	7	288	61	39	4.7	1.6	7.4
ribonuclease	8	571	146	75	3.9	1.9	7.6
ribonuclease	-	68	27	15	2.5	1.8	4.5
ribonuclease	-	8	4	3	2.0	1.9	2.7
head	9	1183	238	105	5.0	2.2	11.3

Table 2 - Rendering times