

Kwan-Liu Ma
 Michael F. Cohen
 James S. Painter

Department of Computer Science, University
 of Utah, Salt Lake City, Utah 84112, U.S.A.

Volume Seeds: A Volume Exploration Technique

SUMMARY

Ray-traced volume rendering has been shown to be an effective method for visualizing 3D scalar data. However, with currently available workstation technology, interactive volume exploration using conventional volume rendering is still too slow to be attractive. This paper describes an enhanced volume rendering method which allows interactive changes of rendering parameters such as colour and opacity maps. An innovative technique is provided which allows the user to plant a 'seed' in the volume to rapidly modify local shading parameters. For a fixed viewing position, the user can interactively explore specific regions of interest. Furthermore, a virtual cutting technique with the exploratory seed allows the user to remove surfaces and see the internal structure of the volume. Examples demonstrate these techniques as an attractive option in many applications.

KEY WORDS: Scientific visualization Volume rendering Interactivity

1. INTRODUCTION

Direct volume rendering,¹⁻⁶ has been shown to be a very effective method for visualizing 3D scalar data. In contrast to geometrical surface rendering methods, volume rendering creates an image directly from the volume of data, without constructing surface primitives. By avoiding the generation of intermediate graphics primitives, direct volume rendered images are usually less prone to distortion and artefacts.^{6,7}

A popular approach to direct volume rendering is based on ray casting.⁴ The volume data is stored as a 3D array of sample values (*voxels*). An image is constructed in *image order* by casting rays from the eye, through the image plane and into the volume of data. One ray per pixel is generally sufficient, provided that the image sample density is higher than the volume data sample density.

The data volume is sampled at evenly spaced points along the ray, usually at a rate of one to two samples per voxel. At each sample point on the ray, a colour and an opacity are computed using trilinear interpolation from the data values at each of the eight nearest voxels. The colour is assigned by applying a shading function such as the Phong lighting model.⁸ A colour map is often used to assign colours to the raw data values. The normalized gradient of the data volume can be used as the surface normal for shading calculations. The opacity is derived using the interpolated voxel values as an index into an opacity map.

Sampling continues until the data volume is exhausted or until the accumulated opacity reaches a threshold cut-off value. The final image value corresponding to each ray is formed by compositing, front-to-back, the colours and opacities of the sample points along the ray.

As with other ray-tracing algorithms, ray-traced volume rendering is generally slow and requires recalculation every time the view position is changed. Rendering time grows linearly with the size of the volume and the number of pixels in the image. Some efforts have been made to reduce the cost of ray

casting and resampling along each ray by using progressive refinement and by exploiting the spatial coherence present in the volume.^{1,9-11} Reductions in image generation time of an order of magnitude have been achieved. Nevertheless, rendering is still too slow to be attractive in an interactive setting. Interactive rendering will be required to allow true 'exploration' of the data.

One approach to interactive volume exploration is the use of special purpose hardware.¹²⁻¹⁵ These efforts are focused on the design of hardware accelerators that can bring real time speeds to the volume rendering processes. Hardware accelerators for volume rendering are not yet generally available, so we have concentrated our work on software solutions using existing graphics workstation technology.

Another approach, proposed in Reference 16, uses interpolation techniques to approximate ray-traced images in near real time. A costly initialization involves computing ray-traced volumetric images from 38 preselected view points at different locations on a sphere surrounding the volume data. After the initialization phase, the user is able to visualize the volume from any angle. However, only the view position can be changed interactively. A change of any other rendering parameters requires another costly initialization phase.

An important part of the volume rendering process is to assign colour and opacity maps. Appropriate colour mapping creates more understandable images and enhances features of interest. Opacity mapping allows the user to focus on a portion of the data by making uninteresting portions transparent. Switching from one set of maps to another offers a different view and possibly different insight into the volume of data. Hence it is essential to equip the user with tools that can modify maps freely and quickly.

We have developed an interactive volume exploration method, called volume seeds, that is based on an image-order front-to-back ray-traced volume rendering algorithm.¹ In volume seeds, rendering is based both on ranges of data values and on *geometric locations*. The user is allowed to plant a seed in the

volume through the use of a data slicer, and to use this seed to modify local shading parameters based on a sample point's location relative to the seed. Techniques such as progressive refinement have also been applied to speed up the recalculation for selecting a viewing position. From a static view position, colour, opacity and local rendering parameters can be modified freely and the resulting image can be generated in a few seconds. Local rendering parameters include parameters related to the use of volume seeds, which will be described in later sections. Although the acceleration gained by volume seeds is restricted to a fixed view position, a new rendered image for a different view position can be produced in a few minutes.

Surfaces of interest can be extracted by controlling opacity maps. It is often useful to see structures in the same range of data values obscured by the extracted surfaces. A cut through the volume serves this purpose, particularly in medical imaging. An extra feature offered by volume seeds is the ability interactively to produce images approximating the rendering of a cut volume.

SAMPLE CACHING

To achieve interactive volume exploration, the key requirement is to avoid or to reduce as much as possible the recalculation of ray casting and resampling. In general, any change of the following parameters requires a recalculation:

1. image size or image resolution
2. viewing position
3. lighting model
4. colour map or opacity map
5. local rendering parameters.

For 1, 2 and 3, techniques such as progressive refinement and hierarchical spatial enumeration^{1,9-11} can help the user pick a viewing position quickly, though every time a recalculation of ray casting and resampling must be carried out. On the other hand, by storing partial results at each sample point along each ray we can avoid much of the rerendering

computations for subsequent changes of any parameters in 4 and 5, values modified by the user more frequently. In general, each sample value is obtained by calculating

1. the sample location,
2. the trilinear interpolation between eight surrounding neighbours
3. the lighting model*
4. the color and opacity compositing.

If the view position and lighting parameters are not changed, only item 4 needs recalculation.

Using the front-to-back ray-traced algorithm, the accumulated colour along each ray and thus of each pixel, C_p , can be described by the following formulae:

$$C_p = \sum_{i=1}^n W(i) C_s(V_i) S(i) \quad (1)$$

where n is the total number of samples along the ray, i is a sample index ($i=1$ is the sample closest to the eye), $W(i)$ is the weighting function, depending only on the opacities of the samples and based on the *under* compositing operator of Reference 17, V_i is the scalar data value at sample i , trilinearly interpolated from the eight nearest voxel values, $C_s(V_i)$ is the colour assigned to the interpolated data value at sample i , $S(i)$ is the shading function depending on the light sources, viewer position, etc.

The weighting function is computed by

$$W(i) = \alpha_s(V_i) \left[1 - \sum_{j=0}^{i-1} W(j) \right] \quad (2)$$

where $\alpha_s(V_i)$ is the opacity assigned to the interpolated data value at sample i , and $W(0)$ is set to 0.

We use a Phong lighting model,⁸ $S(i)$ is given by

$$S(i) = \frac{k_d(\mathbf{N}(i) \cdot \mathbf{L}) + k_s(\mathbf{H} \cdot \mathbf{N}(i))^m + k_a}{d(i) + k} \quad (3)$$

where k_d is the fraction of diffuse reflection, k_s is the fraction of specular reflection, k_a is the fraction of ambient reflection, m is the exponent used to approximate highlight, $d(i)$ is the distance from the image plane to the sample location i , k is the constant used for depth cueing, $\mathbf{N}(i)$ is the surface normal at sample location i , \mathbf{L} is the normalized vector in the direction of the light source, \mathbf{H} is the normalized vector in the direction of maximum highlight

$$\mathbf{H} = (\mathbf{E} + \mathbf{L}) / |\mathbf{E} + \mathbf{L}|$$

and \mathbf{E} is the normalized vector in direction of the viewer.

The surface normal, $\mathbf{N}(i)$, is the normalized gradient, interpolated from the eight nearest sample locations.

According to our previous discussion, $S(i)$ is an invariant provided that the view position and the lighting model remain fixed. During each rendering for new colour, opacity map-

ping or local rendering parameters, recalculation is only required for equations (1) and (2). By caching the interpolated data values V_i and the evaluated lighting model $S(i)$ we can bypass both the lighting model evaluation and the trilinear interpolation computations.

As a result, values for a ray that must be stored include

1. the starting point of each ray
2. the interpolated data value at each sample point, V_i
3. the local shading calculation, $S(i)$.

It is important to note that when caching ray samples, a ray should not be terminated adaptively by the accumulated opacity value. Each ray must be sampled until the volume is exhausted, so that in subsequent interactive re-rendering the opacity map can be arbitrarily modified by the user.

In our current implementation, the location where each ray begins is stored as three floating point numbers for the x , y and z coordinates. If the sample interval is a constant, and we know where each ray starts and the direction vector of the viewer, we can recover each sample point in an incremental manner during subsequent re-renderings. At each sample point, a data value is calculated by interpolating eight surrounding voxels. This value is stored in one byte. $S(i)$ is stored in two bytes to preserve accuracy as much as possible.

As an example, if an image of 240×240 pixels is used to explore a $128 \times 128 \times 128$ volume data set, with a sampling frequency of 1 sample/voxel, a cache of $(240 \times 240 \times (12 + 128 \times 3)) = 23$ MB is needed, which is a manageable size in practice. Though the 240×240 image size may seem small, it can be magnified to fit the size of the available display and provides adequate detail for interactive exploration. Progressive refinement can be used when an image of higher quality is required for further observation. That is, an initial image of 240×240 is computed and displayed with a 2×2 pixel magnification. The 240×240 ray samples are stored for later use. Thereafter, finer images at higher resolutions are formed incrementally. The ray samples collected in the refinement step are not stored, though they could be if sufficient memory were available. An overlay scheme, described in a later section, makes incremental rendering more accessible.

VOLUME SEEDS

In conventional volume rendering, surface normals at each voxel are approximated by the local gradient of the data, which can be computed from either 6 or 26 neighbouring voxel values.¹⁸ Both colour and opacity values are mapped from voxel values. Consequently, the resulting rendered image reflects the data values of interest. A now familiar example is in medical imaging from CT (computed tomography) data, for instance the upper left image in Figure 8. If one is interested in the bone structure, opacity values are set to make soft tissue transparent and bone opaque. Although this creates effective images, it does not allow one to see interior bone structure hidden by an outer structure such as the skull. Frequently, the user wants to focus on a geometrical region of interest in addition to a value range of interest. Volume cutting is a useful tool for this purpose. However, cut-

new rendering must be performed for a new cut volume.

Ideally, the user should be allowed interactively to point to a region in the volume, and a rendered image featuring the region of interest should be created immediately. A good substitute is a volume slicer, which provides a 2D rendering of a slice through the volume. The drawback of a slicer is that it provides limited information, a 2D plane at a time. Since slices can be generated very quickly, slicing is a good first step for data exploration. Speray¹⁹ describes how a slicer can be effectively used to build a mental image. Here we also use a slicer for obtaining an accurate geometrical location to plant a 'seed'. By slicing through the volume, the user sees the full range of data in a particular slice of the volume. Using a mouse, a 2D location on a slice is selected yielding a 3D location within the volume. The slicer then passes this location as a seed to the volume rendering program.

Local shading modification

The seed selected can be used to modify local shading parameters. Surfaces at a large distance away from the seed can be made transparent by modifying the opacity map based on distance from the seed. The seed is used implicitly to define a *matte volume*, as described in Reference 6, which is used to accentuate data near the seed point, s , and attenuate data away from the seed point. The resulting image enhances an area, usually a spherical region, around the seed. Currently three variables are controllable by the user. They are

1. r : radius from the seed
2. mn : minimum matte value
3. mx : maximum matte value

A possible definition for the matte function at a sample point p , $\gamma(p)$ in terms of r , mx and mn is

$$\gamma(p) = mn + (mx - mn) \beta(\text{dist}(p,s), r)$$

$$\beta(d,r) = \begin{cases} \cos^2\left(\frac{\pi d}{2r}\right), & \text{if } d < r \\ 0, & \text{otherwise} \end{cases}$$

where $\text{dist}(p,s)$ is the distance between the seed s and the sample point p in three-space. Figure 1 gives a graphical illustration of these variables.

In essence, r is used to control how wide an area the user wants to see. Surface outside this area should be semi-transparent or fully transparent, determined by mn ; mx is used to indicate how much enhancement is to be made to the area near the seed.

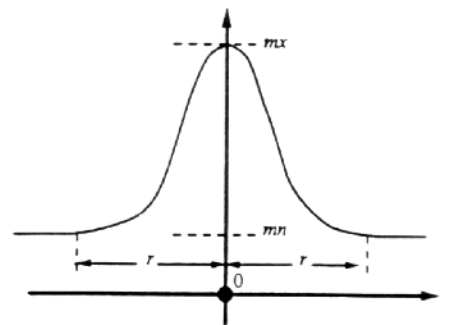


Figure 1. Slice through matte function

*In Reference 4 the lighting model is evaluated once at each voxel rather than at each sample point along the ray. This increases the storage required but speeds up subsequent image generations provided that the lighting model need not be re-evaluated.

Applying $\gamma(p)$ to equation (1), equation (2) becomes

$$W(i) = \gamma(p_i) \left\{ \alpha_s(V_i) \left[1 - \sum_{j=0}^{i-1} W(j) \right] \right\} \quad (4)$$

where p_i is the three-space point corresponding to the i th sample along the ray.

The resulting matte volume can also be applied to colour saturation. Thus the colour at a sample point away from the seed is desaturated.

Overlay rendering

When exploring with volume seeds, if mn is unchanged, recalculation time can be further reduced by using an overlay scheme. This is analogous to exploiting frame coherence in computer animation. Each time a new seed is selected, only the local screen area within the bounds set by r is recalculated. A precomputed image ($mx = mn$) is stored and copied to the screen before the recalculated area is painted. With overlays, we can better take advantage of progressive refinement since now only a small area of the image needs refinement. Sample caching together with volume seeds provide the user with a fast interactive mean of exploring volume data sets from a static view position.

Virtual cut-aways

Opaque surfaces hide the surfaces behind them. Sometimes it is desirable to 'cut away' parts of the opaque material that obscure internal structures in the volume data. The plane of the slicer, used for selecting the seed, can be used as a cutting plane. A dot product of the vector normal to the cutting plane and the position vector from the seed to the sample point can determine whether a sample point is in front of or behind the cutting plane. For example, in Figure 2, p_1 is in front of the seed and p_2 is behind the seed because

$$U_1 \cdot P < 0 \text{ and } U_2 \cdot P \geq 0$$

where U_1 and U_2 are position vectors from the seed to p_1 and p_2 , respectively, and P is the vector normal to the cutting plane. The opacity matte, $\gamma(p)$, is used directly for all points behind the cutting plane. Points in front of the cutting plane are attenuated according to their distance from the cut-off plane.

$$\gamma_{cut}(p) = mn + (mx - mn) \beta(\text{dist}(p,s), r) / \beta_{cut}(\text{dist}(p,\text{cut}))$$

where $\text{dist}(p,\text{cut})$ is the (signed) perpendicular distance from the point p to the cut plane; the sign is negative in front of the cut plane and positive behind. β_{cut} specifies the attenuation factor in front of the cutting plane. One choice for β_{cut} is the unit step function. However, the function's discontinuity may introduce image artefacts. A choice that avoids this problem is

$$\beta_{cut}(d) = \begin{cases} \beta(-d, r_{cut}) & , \text{ if } d < 0 \\ 1 & , \text{ otherwise} \end{cases}$$

which ramps smoothly from 0 to 1, over r_{cut} voxel cells. A reasonable choice for r_{cut} is 2 voxel cells. Figure 3 shows a one-dimensional slice of γ_{cut} , perpendicular to the cutting plane. Note that when $mn > 0$ the data in

front of the cutting plane is not completely eliminated, only de-emphasized, which is why we call it virtual cut-aways.

In addition, if γ_{cut} is made to depend on the data value at the sample point, selective cutting of the volume is possible. This is useful when multiple objects exist in the volume and only certain objects are to be removed. Taking a CT data set as an example, we can cut away tissue in the front half of the volume and leave bone and the complete back half of the volume.

A VOLUME EXPLORATION SYSTEM

We have implemented a prototype system of a volume explorer with the features described above. At present, the system consists of a volume renderer, a data slicer, a colour map maker and an opacity map maker. Each of them runs as an independent process communicating with others via files. Figure 4 shows how each process is related to the others. The system has been designed to be used by people without extensive knowledge of computer graphics or image processing. Figure 5 shows the user interface of the volume renderer (right) and the data slicer (left).

The data slicer is used to select seed points for rendering. The data volume can be

rotated and translated through mouse control. The slicer displays the data slice at a fixed offset from the eye point using the current colour map. During interactive positioning, a low resolution slice is displayed in real time. Once interaction has stopped, the full resolution slice is displayed. A seed point may be selected from the current slice with the mouse.

Local rendering parameters controlling volume seeds can be modified through slider bars on the volume renderer control panel. A cut-away view can be selected by depressing the Cut Away button. The cutting plane is the plane of the slicer. Other controls allow the user to specify whether to cache the sample rays, desaturate colours using the opacity matte, use the opacity matte to enhance opacities, use overlay rendering and use progressive refinement. Depressing the Render button will compute and display the volume rendered image using the existing settings of all the controls. If samples have been cached and the view position has not changed, the new image will appear rapidly, within a few seconds. The system automatically determines whether cached sample data is still valid with the current control settings. Depressing the Show Mapping button will show the current colour and opacity maps.

In Figure 6, both the colour map maker (bottom) and the opacity map maker (top) are shown. The colour map maker allows the user to construct colour maps by interpolating between user assigned key colours. The user can construct opacity maps with splines, as illustrated in the Figure, or by using parametrized templates. Changes to the colour map will be immediately reflected on the display of the data slicer (top left). The user must depress the Render button to see the changes of either the colour map or the opacity map in the volume renderer display. Of course, if ray samples were cached, the resulting rendered image can be generated at once. Figure 6 shows the state after a colour map change but before the Render button is selected.

Currently, the system runs on Silicon Graphics IRIS workstations and IBM RS6000 workstations. The user interface was implemented using Silicon Graphics' graphics library, GL. The system can be ported to most other graphics workstations by replacing the GL interface with an X Window System interface using a toolkit such as Motif. However, in order to use sample caching, main memory should be large enough to hold cached ray samples. Exploration of a volume at a higher resolution, which may require more than 100 MB of memory, is possible by distributing volume rendering to a network of workstations.

RESULTS

The practical value of volume seeds is best shown with an on-screen demonstration. Volume seeds is an effective technique for visualizing 3D scalar volume data of complex structures.

The images in Figure 7 use MRI (magnetic resonance imaging) knee data from the UNC Chapel Hill volume data sets. The original data size was $256 \times 256 \times 127$ 16-bit values which were rescaled and requantized to $128 \times 128 \times 64$ eight-bit values. The upper left image shows a slice through the volume using the data slicer. The upper right shows a normal volume rendered image. It is

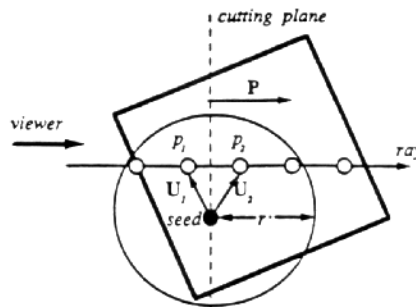


Figure 2. Cutting plane

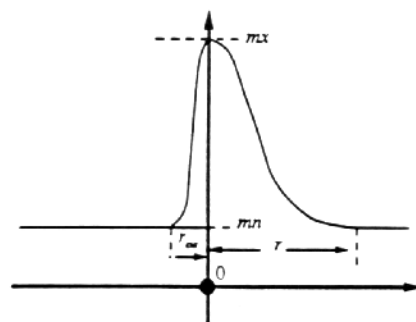


Figure 3. Slice through cut-away matte function

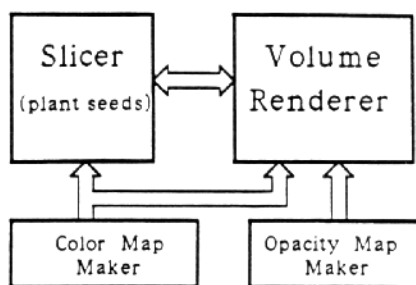


Figure 4. Volume seeds system

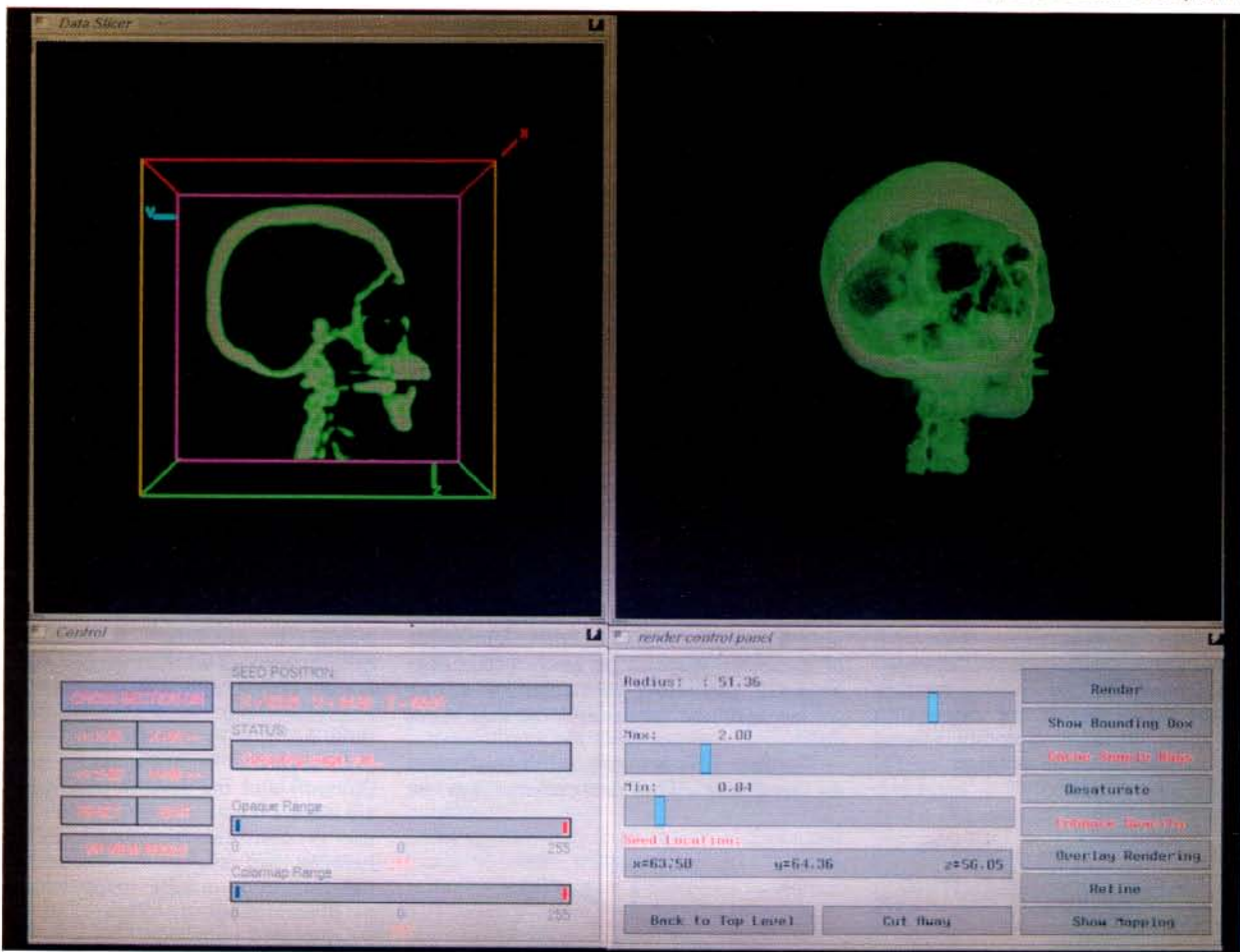


Figure 5. User interface of the volume renderer and slicer

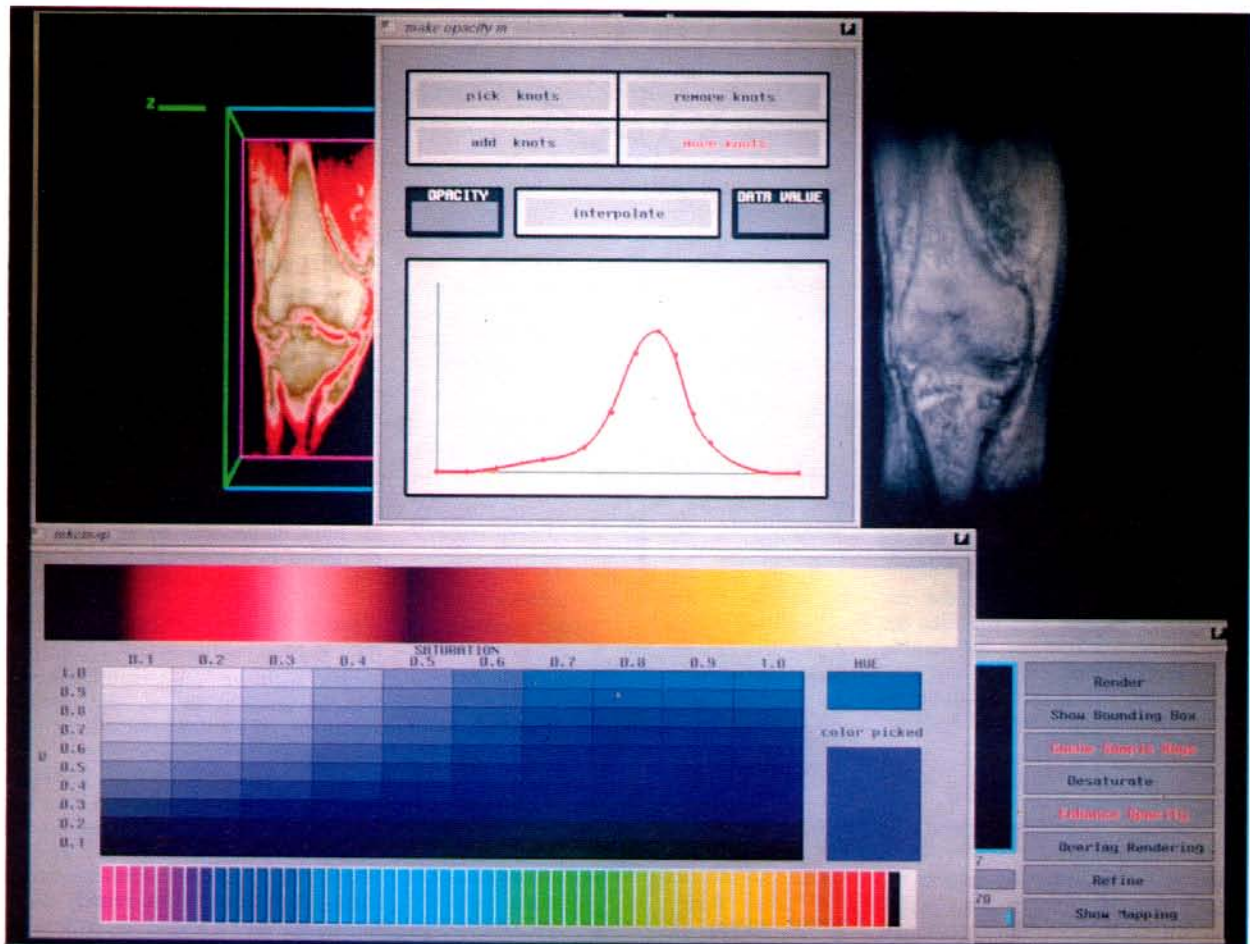


Figure 6. User interface of the colour and opacity map makers

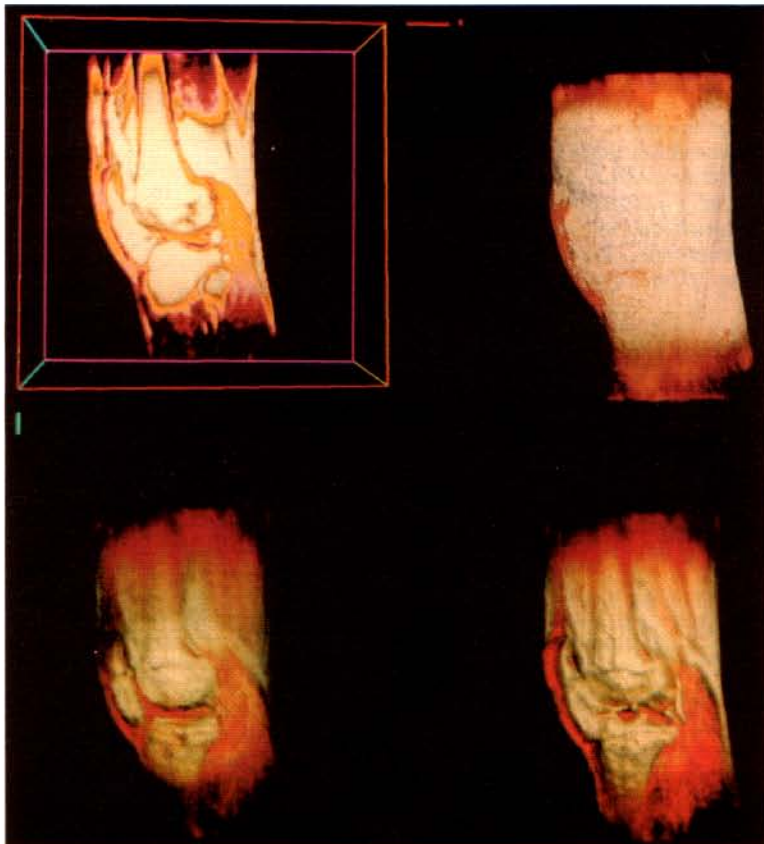


Figure 7. Visualization of MRI knee data

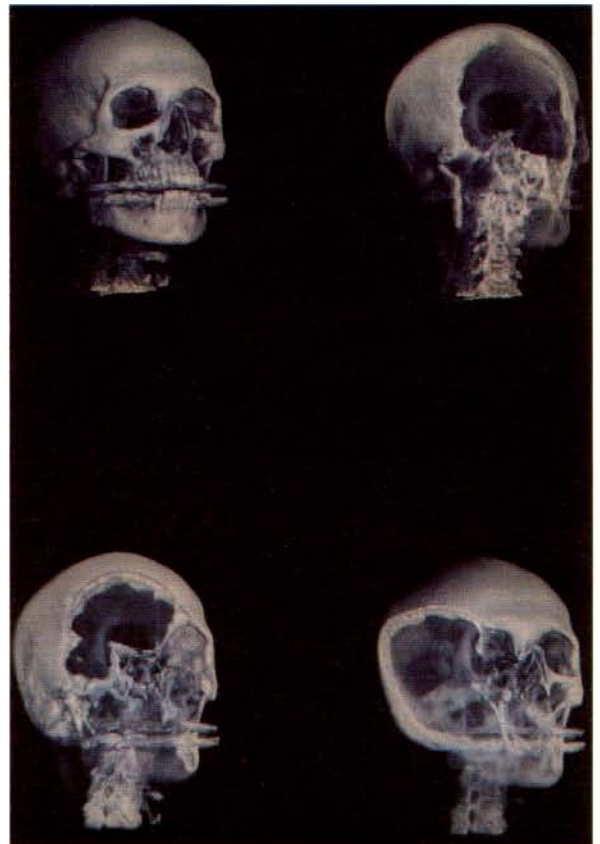


Figure 8. Visualization of CT head data

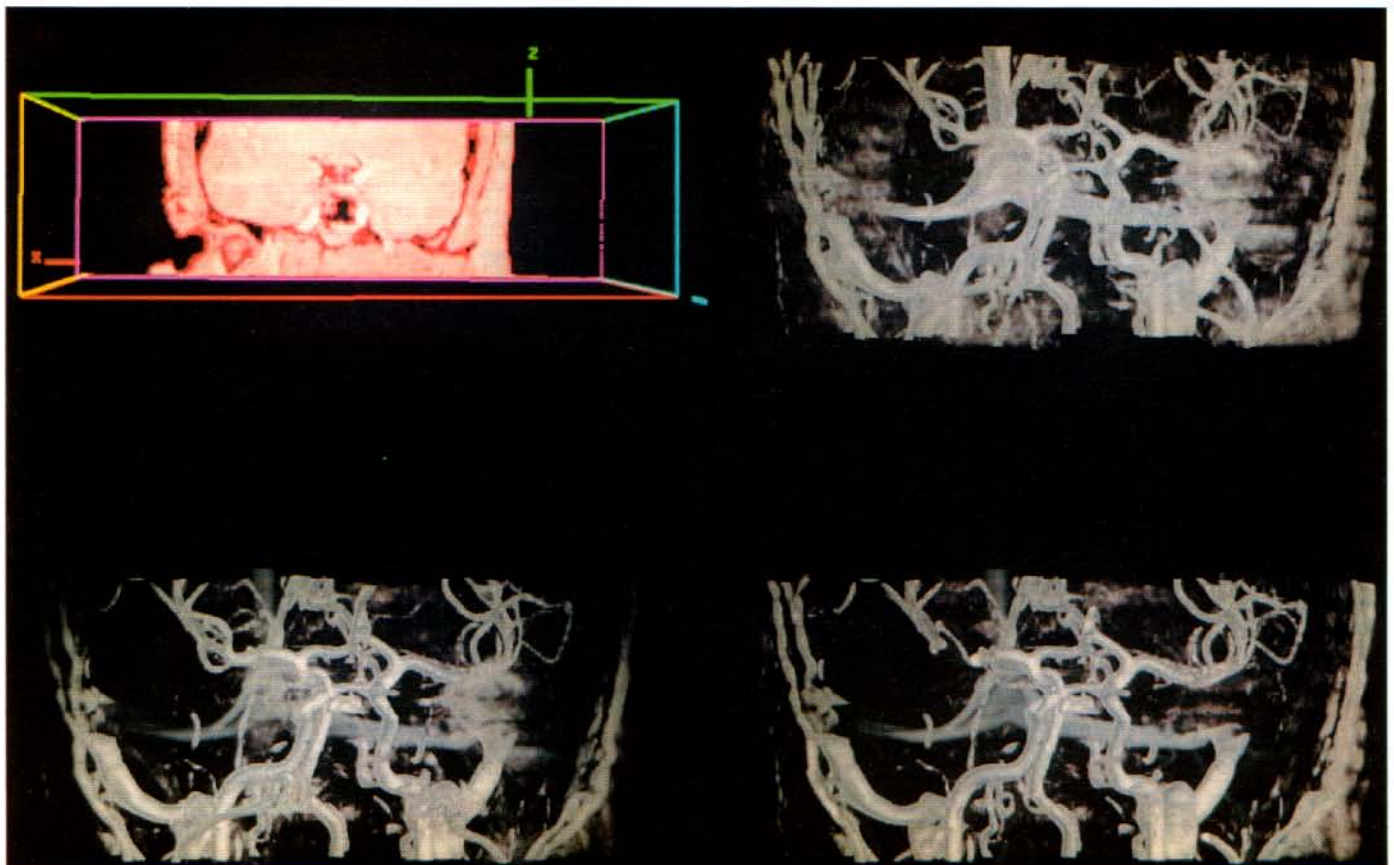


Figure 9. Visualization of MRA vessel data

difficult to see the details of the internal structure because of occluding material in the same data value range, typical of MRI data. The lower left image shows a volume rendered image using a seed placed at the knee joint. Comparing it to the upper right, note that the internal structure is now visible around the seed point. In the lower right image, the same volume rendering as in the lower left one is shown, but with a cut-away, further enhancing detail near the seed point.

Figure 8 shows a set of images using a CT head data set also from the UNC Chapel Hill volume data sets. The original data size was $256 \times 256 \times 113$ 16-bit values, rescaled to $128 \times 128 \times 113$ eight-bit values. The upper left image shows a normal volume rendered image. The other three show the volume rendered image using different seed locations and cut vectors. Note that the seed and cut-aways allow the user to explore the detailed interior bone structure in different areas of the volume.

Finally, Figure 9 depicts the use of volume rendering and volume seeds with MRA (magnetic resonance angiography) data. The data set used, provided by the Medical Image Research Laboratory in the University of Utah, is of size $256 \times 256 \times 78$. It reveals blood vessels in the middle portion of a human head. Using a slicer, as shown in the upper left image, it is very difficult to distinguish vessels from tissue. With normal volume rendering, as shown in the upper right image, 3D vessel structures can be clearly identified. However, the complexity of the vessel structure obscures much of the detail. The lower two images, rendered with two different seed locations, show the added clarity which can be achieved in the neighbourhood of the seeds. Note in particular the small vessels in the centre and lower left regions of the images.

CONCLUSIONS

A method for interactive volume exploration—volume seeds—has been introduced. The method is based on conventional ray-traced volume rendering. Samples along the rays are cached so that subsequent images can be quickly re-rendered when changes are made. The seed technique allows the user to visualize local features efficiently. The

seed location is used to construct a matte function which enhances data near the seed and de-emphasizes data far away from the seed. An overlay scheme, which further reduces the re-rendering time, makes the seed technique even more attractive. Virtual cut-aways are also implemented using the seed point and matte volume. This gives the user increased flexibility in visualizing local features. Volume seeds makes interactive volume exploration possible with current available workstation technology.

ACKNOWLEDGEMENTS

The authors would like to thank members of the Scientific Visualization group and the Alpha_I group at the University of Utah for their support and many enlightening discussions. Elena Driskill implemented the slicer. Jim Cobb at IBM provided many valuable suggestions. Thanks also go to Robert McDermott and James Rose at the Utah Supercomputing Institute. Thanks to Marc Levoy for providing the insight needed to interpret the UNC CT head data. Additional volume data sets were provided by Michael Pique at the Scripps Clinic Molecular Biology and Allen Sanderson at the Medical Imaging Research Lab. This work has been supported in part by an IBM grant for Scientific Visualization under the direction of Richard Riesenfeld, Elaine Cohen and Ronald Pugmire.

REFERENCES

1. M. Levoy, 'Efficient ray tracing of volume data', *ACM Transactions on Graphics*, **9**, (3), 245-261 (1990).
2. C. Upson and M. Keeler, 'V-buffer: visible volume rendering', *Computer Graphics (Proceedings of SIGGRAPH 1988)*, **22**, (4), 59-64 (1988).
3. P. Sabella, 'A rendering algorithm for visualizing 3D scalar fields', *Computer Graphics (Proceedings of SIGGRAPH 1988)*, **22**, (4), 51-58 (1988).
4. M. Levoy, 'Display of surfaces from volume data', *IEEE Computer Graphics and Applications*, May 1988, pp. 29-37.
5. J. Kajiya and B. Herzen, 'Ray tracing volume densities', *Computer Graphics (Proceedings of SIGGRAPH 1984)*, **18**, (3), 165-174 (1984).
6. R. A. Drebin, L. Carpenter and P. Hanrahan, 'Volume rendering', *Computer Graphics (Proceedings of SIGGRAPH 1988)*, **22**, (4), 65-74 (1988).
7. M. Levoy, 'A taxonomy of volume visualization algorithms', *SIGGRAPH 1990 Course Notes 11, Volume Visualization Algorithms and Architectures*, August 1990.
8. B. Phong, 'Illumination for computer-generated pictures', *CACM*, **18**, (6), 311-317 (1975).
9. K. Subramanian and S. Fussell, 'Applying space subdivision techniques to volume rendering', *Proceedings of Visualization 90*, October 1990, pp. 150-159.
10. R. Meyers and M. Stephenson, 'Ray traced scalar fields with shaded polygonal output', *Proceedings of Visualization 90*, October 1990, pp. 263-272.
11. M. Levoy, 'Volume rendering by adaptive refinement', *Visual Computer*, **6**, (1), 2-7 (1990).
12. A. Kaufman and R. Bakalash, 'Memory and processing architecture for 3D voxel-based imagery', *IEEE Computer Graphics and Applications*, **8**, (6), 10-23 (1988).
13. A. Levinthal and T. Porter, 'Chap—a SIMD graphics processor', *Computer Graphics (Proceedings of SIGGRAPH 1984)*, **18**, (3), 77-82 (1984).
14. H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs and L. Israel, 'Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories', *Computer Graphics (SIGGRAPH '89 Proceedings)*, **23**, (3), 111-120 (1989).
15. M. Levoy, 'Design for a real-time high-quality volume rendering workstation', *Proceedings of the 1989 Chapel Hill Workshop on Volume Visualization*, May 1989.
16. T. Foley, D. Lane and G. Nielson, 'Towards animating ray-traced volume visualization', *The Journal of Visualization and Computer Animation*, **1**, 2-8 (1990).
17. T. Porter and T. Duff, 'Compositing digital images', *Computer Graphics (Proceedings of SIGGRAPH 1984)*, **18**, (3), 253-259 (1984).
18. U. Tiede, K. Hoehne, M. Bomans, A. Pommert, M. Riemer and G. Wiebecke, 'Investigation of medical 3D-rendering algorithms', *IEEE Computer Graphics and Applications*, March 1990, pp. 41-53.
19. D. Speray, 'Volume probes: interactive data exploration on arbitrary grids', *Computer Graphics (Workshop on Volume Visualization)*, **24**, (5), 5-12 (1990).