EUROGRAPHICS '96 / J. Rossignac and F. Sillion (Guest Editors), Blackwell Publishers

Volume 15, (1996), Number 3

Incremental Volume Rendering Using Hierarchical Compression

Michael B. Haley and Edwin H. Blake

Department of Computer Science, University of Cape Town, Private Bag, Rondebosch 7700, South Africa.
Email: mikeh@aztec.co.za ; edwin@cs.uct.ac.za

Abstract

We present a new algorithm here for efficient incremental rendering of volumetric datasets. The primary goal of this algorithm is to give average workstations the ability to efficiently render volume data received over relatively low bandwidth network links in such a way that rapid user feedback is maintained. Common limitations of workstation rendering of volume data include: large memory overheads, the requirement of expensive rendering hardware, and high speed processing ability. The rendering algorithm presented here overcomes these problems by making use of the efficient Shear-Warp Factorisation method which does not require specialised graphics hardware. However the original Shear-Warp algorithm suffers from a high memory overhead and does not provide for incremental rendering which is required should rapid user feedback be maintained. Our algorithm represents the volumetric data using a hierarchical data structure which provides for the incremental classification and rendering of volume data. This exploits the multiscale nature of the octree data structure. The algorithm reduces the memory footprint of the original Shear-Warp Factorisation algorithm by a factor of more than two, while maintaining good rendering performance. These factors make our octree algorithm more suitable for implementation on average desktop workstations for the purposes of interactive exploration of volume models over a network. Results from tests using typical volume datasets will be presented which demonstrate the ability of the algorithm to achieve high rendering rates for both incremental rendering and standard rendering while reducing the runtime memory requirements.

Keywords: incremental, volume rendering, octree, Shear-Warp

1. Introduction

With the rising popularity of the Internet and the ever increasing range of data available through systems such as the World-Wide-Web, new methods of transferring and displaying various data types become necessary. Specifically, volumetric datasets, such as those produced by MRI or CT scanning methods, will be available for interactive exploration.

Due to the extreme size of these datasets their reception over a low bandwidth network connection is time consuming. This has lead previous researchers only to transmit static images [1], and to perform all the rendering on the server. However in a World-Wide-Web environment the amount of potential simultaneous accesses to this data could be very high, and it would then be more efficient to perform the rendering on the client's workstations. This of course requires the transmission of the entire volume dataset to the client and given that this is a time consuming operation, it would be necessary for the client to render the data incrementally as it arrives in order to achieve reasonable user feedback.

Our algorithm builds a compressed hierarchical representation of the volume data using an octree which may then be transmitted to a client (Section 3). On reception of this data the client may incrementally classify (Section 4) and render the data (Section 5) using a modified Shear-Warp Factorisation algorithm. Due to the fact that the incremental classification and rendering times are in the order of seconds, while the network transmission times are in the order of 10 minutes or more, multiple incremental renderings may be performed of the data as it arrives, giving the user the ability to rapidly examine overall characteristics of the data. The result of our method is a rapid incremental volume renderer which reduces the memory requirements of previous direct volume rendering algorithms.

2. Prior Work

Our algorithm falls into the *lossless* compression category (i.e. all information content of original is maintained) and is also notable for its ability to render the data in its compressed state, therefore not incurring the decompression overhead.

An early lossless and highly used approach is that of *run-length encoding* of the data. Montani and Scopigno [2] defined the STICKS representation scheme, which performed run-length encoding along a set axis (typically the z axis). While providing better overall compression ratios than a pure octree method it is unfortunately not symmetrical in 3-space and therefore the memory access patterns change dramatically depending on the traversal order in the volume. Wilhelms and Van Gelder [3] have performed a lot of work in the area of hierarchical data representation using variants of the basic octree method. Their research has indicated that hierarchical representation provides flexibility, and while not providing optimal compression, allows the compressed version to be used almost as quickly as the original as well as providing the ability to naturally represent the data in reduced accuracy. Ning and Hesselink [4] propose the use of vector quantization on blocks of voxels to generate a compressed volume which may be rendered in its compressed state. Their scheme does not however maintain the original content exactly and their rendering rates are fairly low. Other approaches include the *truncated Huffman coding* used by Fowler and Yagel [5], and the DCT based scheme of Yeo and Liu [6], but in both cases there is no ability to render the data in its compressed state.

The rendering of volumetric datasets historically consisted of isosurface methods such as Lorensen and Cline's [7] *Marching Cubes Algorithm*, and our *Mesh Propagation Algorithm* (Howie and Blake [8]). Recent work has concentrated on direct volume rendering, with approaches such as ray-casting and splatting methods. Laur and Hanrahan [9] outline an algorithm for progressive refinement rendering of volume data using hierarchical splatting. Their algorithm, while also using octrees, makes use of dedicated polygon rendering and compositing hardware to achieve efficiency.

Incremental rendering, or rendering at lower resolutions, has also been achieved in the past through frequency space manipulations of the data. Muraki [10] and Westermann [113 propose a wavelet decomposition of the data, which allows for a multiresolution representation of the data and Totsuka and Levoy [12] propose a frequency domain volume renderer. These methods, while allowing for facilities such as edge enhancement and reducing run-time data size, are still too complex to be considered for implementation on average desktop workstations.

Most recently the Shear-Warp rendering algorithm was presented by Lacroute and Levoy [13], which allows for near real-time rendering on an average workstation. Their algorithm makes use of a run-length encoding technique similar to the STICKS scheme mentioned above, except that the compression must be in a direction orthogonal to the viewing direction. Due to the dependence of the compression direction on viewing angle, three versions of the compressed volume have to be maintained in memory to achieve acceptable rendering rates.

In order to achieve an incremental rendering scheme the data has to be represented in some hierarchical fashion where basic structure is first transmitted followed by refinements to the basic structure. Our algorithm implements the Shear-Warp rendering approach using an octree representation of the data which

removes the need to store three versions of the volume (due to the symmetry of the octree) while not having any significant impact on rendering time. The algorithm then makes use of the implicit structural information contained in the octree to provide the ability to render approximated areas of the volume when the physical data is not available. Unlike Laur and Hanarahan's [9] method, error terms are not used to determine which nodes in the octree to render as the objective here is to achieve the best possible image. This implies the use of the lowest possible nodes in the octree.

3. Compression of Volume Data

In this section we deal with the initial compression of the data and the construction of the hierarchical data structure. The next two sections will cover the incremental phases of classification and rendering of the volume.

Data Structure

The choice of data structure for the nodes in the octree is critical in achieving efficient traversal and in providing enough information to facilitate approximate rendering of entire sub-trees. Each node structure contains the following:

- An *empty* flag (denoting a region with all transparent voxels), afull flag (denoting a region which is not further subdivided as it is mostly opaque), or a child node reference (which references another eight nodes).
- A raw data reference (if leaf-node), as well as minimum, maximum, and average voxel values. Also the values of the eight comer voxels.
- Minimum, maximum, and average spatial gradients, as well as the average normal vector.

One of the many advantages of the octree representation over the run-length encoding (RLE) representation is that the octree data structure contains references into the raw volume data and is completely separable from it. This implies that different octree structures may be used at the various processing stages. (e.g. It is not necessary to transmit the minimum and maximum gradient fields as they are calculated during classification.) By using a different octree structure during and after classification the transmitted data can be reduced.

Our compression algorithm has four main stages: (1) correction of range of data values and reduction to 8-bit data; (2) initial construction of octree; (3) reordering of octree; and (4) compression of leaf nodes.

Algorithm

In the first stage the three dimensional input array of scalars is converted into an array of eight bit values by selecting the range of "usefil" values in the original dataset and then mapping these values into an eight bit range. This process is akin to the *contrast stretch* algorithm used in image processing. The initial range is selected by heuristics, working with the histogram of the original data, and is then refined through user interaction.

In the second stage, minimum and maximum threshold values are selected which allow the construction algorithm to discard data which is not relevant (e.g. air spaces, or filtering out of low-density tissue). The construction process begins by recursively subdividing the volume into eight sub-volumes. At each level, if all child nodes either all contain relevant data or are all empty, these nodes are gathered together into one parent node. The leaf nodes then either reference the raw data for their sub-volumes or have flags marking them as empty. All higher nodes in the octree store the averaging information for the nodes below.

Thirdly, once the octree is constructed, the nodes which are stored in a linear array in memory are then reordered such that nodes higher up in the octree occur before nodes further down in the octree. This is to allow for the incremental transmission of nodes.

Finally, to produce more efficient compression during the impending transmission of this data, the data at the leaf nodes is compressed using run-length encoding and is stored in the order of node occurrence after the array of node structures. It should be noted however, that this node compression is only used during the transmission of the volume, and it should be decompressed on reception if the advantage of using the symmetrical octree data structure is to be maintained.

Results

Tests were performed at a variety of minimum threshold levels (the maximum threshold level was always set to the maximum value), for both our octree method and the pure run-length encoding method. The maximum octree depth was set to 6 in all cases as this was found to give optimal memory and processing performance. The datasets used are typical medical datasets as well as an engine block dataset (their sizes are given in the table below). Table 1 presents both the resultant data size (in bytes) after compression using the octree method (in bold) and the size after pure run length compression for each test case.

Table 1: *Comparison of compression ratios for octree and RLE methods. (Octree results are in bold.)*

Minimum Level	MR Brain (256x256x 109)		CT Head (256x256x110)		MR Knee (256x256x 127)		Engine (256x256x113)	
10	**2322480**	2075681	**813611**	751408	**6168847**	6099703	**2059335**	1941438
20	**1721950**	1704860	**653220**	621771	**3019784**	2797554	**1630027**	1559309
60	**1333882**	1296588	**499543**	483123	**1516436**	1419247	**1430877**	1357717
130	**176581**	221797	**257632**	271221	**544621**	509277	**1135951**	1052758
200	**20461**	118018	**17261**	123881	**15921**	133856	**93729**	167825

From these results it is clear that the octree algorithm achieves compression ratios very close to those of pure run-length encoding even though an entire octree data structure containing averaging information is included in the octree data sizes. (Note these values do not represent the run-time memory sizes of the algorithms during rendering.) The duration of the entire octree construction process is in the order of 15 seconds, but this is irrelevant in the entire system as this operation only needs to be performed once by the volume server.

4. Classification

Background

Classification of volume data primarily requires the estimation of a spatial gradient at every voxel and the calculation of a normal vector. This process is generally performed using finite differences between sets of neighbouring voxels. In our octree scheme this poses a problem (we use first order differences) as the referencing of voxel neighbours can be time consuming (due to a tree traversal) when a voxel's neighbour lies in a different octree node to itself. Our algorithm alleviates this problem through the use of a octree node caching mechanism and an efficient cache search method. (See below.)

The classification stage also involves the elimination of transparent areas of the volume given an opacity transfer function. (This function takes a voxel's parameters and returns an opacity.) In the original Shear-Warp Factorisation method a *Min-Max* octree was constructed and used in conjunction with a summed-area table to make the classification process more efficient In our algorithm the octree data structure already closely resembles the *Min-Max* octree as it already contains the maximum and minimum values for the various voxels at any given node. Thus the overhead of constructing a *Min-Max* octree is eliminated using our octree method. Another advantage of the octree data structure is that the classification process may flag an entire sub-tree as transparent thus not referencing it at all during rendering.

Caching Mechanism

To address the problem of referencing across octree nodes a caching mechanism is used. The cache simply keeps the last N nodes (where N is the size of the cache) accessed and the co-ordinates of the limits of the nodes. The cache is implemented as follows:

An array A of size N is defined, where each entry contains a hash value h which is generated by bit interleaving the (x,y,z) values which are the co-ordinates of the node. The bit interleaving method is outside the scope of this paper and details can be found in [14]. Each entry also contains: the dimension d (always a cube) of the node, an LRU (least recently used) count for that array cell, and a reference to the node in the octree corresponding to the co-ordinate (x,y,z) and dimension d.

The array is then sorted by the hash value h. A standard LRU cache system is then implemented using the value h as the search key. A matching h value is necessary but not sufficient so a further check of the co-ordinate against the dimension d of the node is necessary to validate that it lies within the node.

The optimal cache size was empirically determined to be 7, however in practice further reclassifications of the data are still fairly time consuming as they cannot be amortised over the transmission times.

Incremental Algorithm

Due to incremental transmission, certain areas of the octree could be missing during classification. Thus when an access to a neighbouring node is required to classify a voxel, there may be insufficient data. Our algorithm overcomes this problem through the use of a single-bit flag in each classified voxel indicating whether it has been correctly classified. Whenever a classification is performed on incomplete data the following process is then followed:

1. If data exists for node then classify all non-edge voxels.
2. Attempt to classify all edge voxels by visiting neighbouring nodes.
3. If a neighbouring node does not yet exist then set the bit flag, and set the classification parameters for that voxel to approximate values.
4. When a neighbouring node is located the bit flag for the adjacent voxel is checked, and if set that voxel is classified as well.
5. Once a successful classification is performed on any voxel the bit flag is cleared.

Results

Table 2 presents the full volume classification times over the four test volumes for the octree compressed data and the RLE compressed data.

Table 2: *Comparison of classification times. (All times in milliseconds and the octree method times are in bold.)*

Level	MR Brain		CT Head		MR Knee		Engine	
10	**27322**	7856	**30272**	5069	**30393**	12797	**62148**	9779
20	**26264**	7577	**29071**	4845	**29875**	11883	**60395**	9349
60	**23897**	6941	**27718**	4530	**26927**	11183	**58292**	8835
130	**22314**	6559	**25890**	4119	**24569**	10572	**54649**	7991
200	**22080**	6511	**24886**	4004	**23991**	10461	**53136**	7778

Although these figures show that the octree classification can in some cases be up to 52 seconds longer than RLE classification, there are factors which are not present in this data:

- The RLE data still has to be transposed into three different volumes which was found to take roughly 8 seconds.
- The transmission time (over a slow network link) could be in the order of 10 minutes or more so the initial classification may be amortised over the transmission time of the volume. This is not possible with the RLE data. The octree overhead is however significant on further reclassifications.

5. Incremental Rendering

Rendering Process

As mentioned in the introduction, the Shear-Warp Factorisation rendering method is used in our algorithm. We progress with the details of our new algorithm, which extends this method for the purposes of rendering partial volume data.

The core of the method relies on the fact that given an overall view transformation matrix $\boldsymbol{M_{view}}$ then,

$$M_{view} = M_{warp} M_{shear-scale} P$$

where $\boldsymbol{P}$ is a permutation matrix which simply reorders the co-ordinates such that the principal viewing axis is always the z-axis. $\boldsymbol{M_{shear\text{-}scale}}$ is a three-dimensional shearing and scaling matrix which shears the slices of the volume in two directions transverse to the viewing direction and scales the slices in the other direction. $\boldsymbol{M_{warp}}$ is a two-dimensional warp matrix which converts the projected object order slices into the resultant image. If the $\boldsymbol{M_{view}}$ matrix is a parallel projection matrix then the scaling will be zero and the warp matrix will be affine. Otherwise if $\boldsymbol{M_{view}}$ is a perspective transformation matrix then the scaling factor will be negative and the warp matrix will be a perspective warp matrix.

The matrix $\boldsymbol{M_{shear\text{-}scale}}$ therefore contains two shearing factors which we will call $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ and a scaling factor $\boldsymbol{q}$. The permutation matrix $\boldsymbol{P}$ is always chosen in such a way that:

$$-1 \le s_x \le 1 \quad \text{and} \quad -1 \le s_y \le 1$$

The values $\boldsymbol{s_x}$, $\boldsymbol{s_y}$, and $\boldsymbol{q}$ are critical in our algorithm as they will dictate the order in which the octree is to be traversed.

One of the primary advantages of the original Shear-Warp Factorisation algorithm was that the volume slices could be composited onto an intermediate image using the front-to-back *over* operator [15]. This allowed the algorithm to efficiently skip large areas of the slices towards the back of the volume when the corresponding areas in the compositing buffer were opaque. When combined with the run-length encoding order of slices a very efficient simultaneous object-order and image-order traversal can be performed which skips both transparent runs in the volume and opaque runs in the compositing buffer.

Our octree algorithm achieves similar results by treating each node in the octree as being a small self contained sub-volume placed at a three-dimensional offset. If a particular sub-volume contains non-transparent voxels then it will be composited onto the intermediate image by using a simultaneous image-order and object-order scan of the data in that sub-volume. However, in order to ensure that correct occlusion is maintained within the volume, the sub-volumes in the octree have to be cornposited in a particular order. This is analogous to the occlusion compatible traversal solution [16] used in rendering height fields and the solution is presented in the next two sections.

Using this traversal method entire regions of the volume may be skipped from processing altogether which makes the octree algorithm perform more efficiently than the run length encoding algorithm when large

amounts of volume data are transparent. A future version of our octree algorithm will incorporate quad-trees into the compositing buffer allowing entire non-transparent sub-volumes to be discarded when the corresponding area of the compositing buffer is opaque.

Parallel Projection Traversal Order

The simpler of the two projection matrices is the parallel projection so we will examine this first.

The shearing factors s_x and s_y completely predict the order of traversal of the octree in the parallel projection case (the scaling factor q is 0 in this case). In order to explore the method of traversal we will consider the reduced case of a two-dimensionally sheared quadtree in the direction s_x. Figure 1 shows the shearing of the quadtree for the three cases: $s_x<0$, $s_x=0$, and $s_x>0$. The numbers indicate an acceptable traversal order in each case, such that front-to-back compositing will be correct (assuming that the one-dimensional image plane is at the bottom).

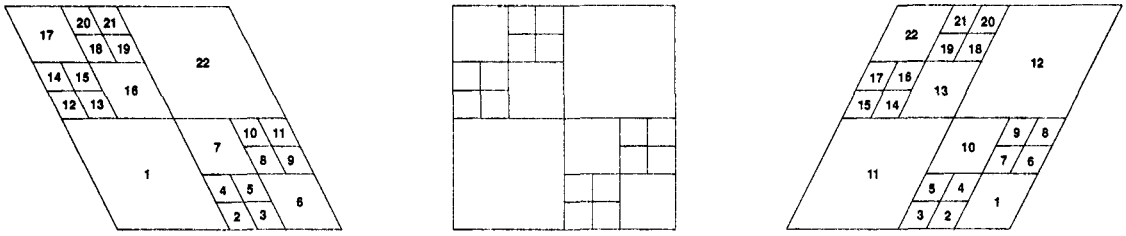

Figure 1. *Three cases of shearing in parallel projection.*

On examination of Figure 1 it is clear that the order of traversing sub-nodes is the same, independent of the location or size of the parent node. This is due to the fact that the shearing amounts are the same throughout the volume. It is therefore sufficient to traverse the tree hierarchically provided that the nodes at each level are visited in a set predefined order which will depend on the shearing factors.

This process is then naturally extended into three dimensions where instead of two possible shearing directions (as with the quadtree above) there are four possible shearing directions corresponding to $(s_x<0, s_y<0)$, $(s_x>0, s_y<0)$, $(s_x<0, s_y>0)$, and $(s_x>0, s_y>0)$. Four possible traversal orders are thus possible.

Perspective Projection Traversal Order

The shearing and scaling matrix resulting from the factorisation of a perspective transformation, contains two shears s_x and s_y as well as a scaling factor q which makes the slices smaller proportional to their distance from the image plane. As before Figure 2 depicts the two-dimensional case of this problem, where there is only a shearing factor s_x and the scaling factor q. Traversal orders are depicted for various values of $s_x>0$ with the scaling factor $q<0$ staying constant.

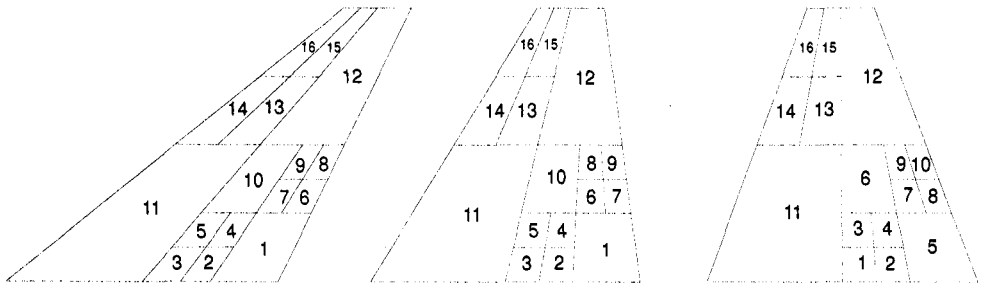

Figure 2. *Three cases of shearing in perspective projection.*

Figure 2 demonstrates that the traversal of the sub-nodes of any one parent node is not the same throughout the quadtree (e.g. nodes *(6,7,8,9)* move around in each case). Instead the traversal depends on whether the slope of the centre line (of the parent node) is negative or positive. In the three dimensional case of the octree the traversal then depends on the slopes of the centre planes of a particular sub-volume.

Given any sub-volume in the octree with origin position **(x,y,z)** in object space and of size ***v*** (the sub-volume is always a cube), then the slope of the x-axis parallel centre plane is

$$Slope_X = zs_x(1+qz)(x+v) - (z+v)s_x[1+q(z+v)](x+v) = -(x+v)s_x[v+zv+vqz+v^2q]$$

Considering that we are only interested in the sign of the slope, and that the x and v values will always be positive, then

$$SlopeSign_X = -s_x(1+z+qz+vq)$$,and for the y-axis parallel centre plane,
$$SlopeSign_Y = -s_y(1+z+qz+vq)$$

Again there is the choice between four possible traversal orders. The choice cannot however be performed once before rendering, but instead has to be made for every octree node when its child nodes are traversed.

When performing a perspective rendering of a volume and the viewing point is very close to the volume or inside it the factorisation can result in more than one primary axis being chosen. In this case the volume has to be broken up into separately rendered sections which causes difficulties with the octree representation. We chose to prevent these cases from occurring by restricting this proximity of the viewing point to the volume.

Averaging Neighbours

The original Shear-Warp Factorisation algorithm used bilinear filtering on individual slices during the compositing process. This was to allow for non-integer shearing and to provide a degree of anti-aliasing. Due to the hierarchical nature of the octree and the difficulty of referencing voxel neighbours across the boundaries of sub-volumes, this bilinear filtering becomes very inefficient in our algorithm.

On comparing experimental results (Plates 4, 5 and 6*compare the use of filtering), we found that the advantages of the filtering process were outweighed by the performance improvement gained by omitting it. In omitting this averaging process, the side effects consist principally of aliasing artifacts as well as a slight sub-pixel distortion of the resulting image. We have found these to be visually acceptable even under animation of the volume. Future work will however concentrate on re-introducing the filtering.

Rendering Non-Leaf Nodes

Due to the incremental nature of our rendering algorithm it is necessary to render areas of the volume when the voxel data is not present. This is achieved by approximating areas of the volume using the values stored in the octree. The higher nodes (larger sub-volumes) will provide very course approximations but as more data arrives lower nodes in the octree (smaller sub-volumes), which approximate the volume more closely, become available. The highest quality of course is achieved by rendering the leaf nodes of the octree.

During the construction and classification the following data is stored in the octree for approximate rendering:

- Voxel values at each comer of the sub-volume.
- The average gradient magnitude of non-transparent voxels in the sub-volume.
- A weighted average of the normals of non-transparent voxels in the sub-volume. This weighting is such that the normals in areas where the gradient magnitude is high have more influence on the average.

* **See page C–55 for plates 4, 5 and 6.**

To calculate the approximate value of any one voxel in the sub-volume, the algorithm uses a trilinear interpolation function with each of the corner voxel values. The formula for a single voxel value V is then:

$$V(x,y,z) = C_0 + xC_1 + yC_2 + zC_3 + xyC_4 + xzC_5 + yzC_6 + xyzC_7$$

where ***(x,y,z)*** is the relative position within the sub-volume and the C_n values (where n=[0,7]) are constants depending on the voxel values at each comer of the sub-volume. Inside the rendering algorithm this function is evaluated inside a third order loop , each level of which corresponds to an unknown in the above equation. Using loop-unrolling, this function can be incrementally evaluated by just using additions and some initial calculations.

This approximated value ***V*** is combined (using the opacity transfer function) with the average surface normal and average gradient magnitude to compute opacity and shading for the voxel so that it may be composited.

Results

Plates 1, 2, and 3" depict the results of rendering the MR Brain volume at various levels of approximation (given by the amount of data available). It is clear that the image quality rapidly improves as lower levels of the octree become available.

Figure 3 shows the results of parallel projection rendering of the MR Brain volume at various threshold levels. The times at each level are calculated by averaging the rendering times of the volume at a variety of rotations. The octree algorithm improves on the RLE algorithm for higher levels as the octree is completely omitting larger areas of the volume. Performance of the octree algorithm is seen to be comparable for both full rendering and partial rendering of the volume. (The partial rendering above is such that every leaf-node in the octree is approximated. This is the worst case scenario.) Using the octree algorithm, perspective rendering was in the order of 5 times slower than parallel rendering. (Perspective results for RLE were not available) In all cases the rendering times were essentially constant independent of rotation angle.

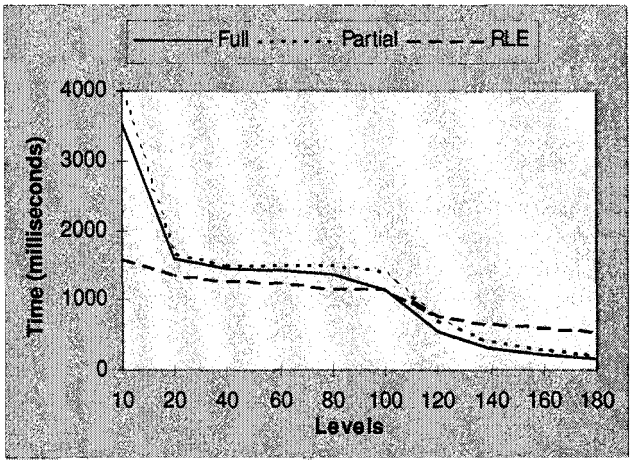

Figure 3: *Times at various levels for full and partial octree rendering compared to normal RLE rendering.*

Table 3: *Runtime memory usage. (Octree method in bold)*

Level	MR Brain	CT Head	MR Knee	Engine
20	**7576452**	**3563576**	**15966684**	**7189112**
	20458320	7461252	33570648	18711708
160	**713980**	**1058644**	**1788604**	**805352**
	2031540	21 10356	2797740	2436144

Finally Table 3, presents the run-time memory usage (in bytes) of each of the algorithms for the four test volumes at threshold levels of 20 and 160. The octree algorithm is a lot more memory efficient.

6. Concluding Remarks

Users of a networked workstation can make use of this algorithm to perform interactive exploration of volumetric data over a typical Internet connection. Where data transmission rates are low, the algorithm provides for rapid incremental rendering of the data as it is received by the workstation. This allows the initial volume classification and rendering to be performed during the transmission period such that efficient user feedback is maintained. The octree rendering algorithm achieves rendering rates which are comparable

* See page C–55 for plates 1, 2 and 3.

with the original Shear-Warp Factorisation algorithm for both incremental and full rendering. This, combined with the fact that the runtime memory footprint of the original algorithm is vastly reduced, makes our octree algorithm very attractive for implementation on workstations.

Although the initial classification is completely amortised over the transmission time, the performance of re-classification is still fairly low so future research will look into using neighbour references in the octree to accelerate this. Other future improvements include using quadtrees for accelerating the compositing buffer and incorporating some form of filtering into the rendering process to reduce aliasing. Compression ratios of the transmitted data may also be improved by using a more complex leaf node compression algorithm such as Huffman coding or vector quantization.

Acknowledgements

We would like to thank the University of North Carolina (Chapel Hill) for making the medical datasets available and Stanford University for making the algorithm and engine datasets available. We also express our gratitude to the South African Foundation for Research Development for funding this research.

References

1. S.A. Cheong, D.C. Martin, and M.D. Doyle. *Integrated Control of Distributed Volume Visualisation Through the World-Wide-Web.* IEEE Visualisation '94 Proceedings, 13-20, 1994.
2. C. Montani and R. Scopigno. *Rendering volumetric data using the sticks representation scheme.* ACM Siggraph, San Diego Workshop on Volume Visualization, 24(5):87-93, 1990.
3. J. Wilhelms and A. Van Gelder. *Multi-Dimensional Trees for Controlled Volume Rendering and Compression.* ACM Siggraph Symposium on Volume Visualization, 27-34, 1994.
4. P. Ning and L. Hesselink. *Fast Volume Rendering of Compressed Data.* IEEE Visualisation Proceedings '93, 11-18, 1993.
5. J. Fowler and R. Yagel. *Lossless Compression of Volume Data.* ACM Siggraph Symposium on Volume Visualization, 43-50, 1994.
6. Y. Boon-Lock and L. Bede. *Volume Rendering of DCT-Based Compressed 3D Scalar Data.* IEEE Transactions on Visualization and Computer Graphics, 1(1):29-43, 1995.
7. C.T. Howie and E.H. Blake. *The Mesh Propagation Algorithm for Isosurface Construction.* Computer Graphics Forum, Eurographics, 13(3):64-74, 1994.
8. W.E. Lorenson and H.E. Cline. *Marching cubes: A High Resolution 3-D Surface Construction Algorithm.* ACM Siggraph Computer Graphics Proceedings, 24(5): 163-169, 1987.
9. D. Laur and P. Hanrahan. *Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering.* ACM Siggraph Computer Graphics Proceedings, 25(4):285-288, 1991.
10. S. Muraki. *Volume Data and Wavelet Transforms.* IEEE Computer Graphics and Applications, 13(4):50-56, 1993.
11. R. Westermann. *Multiresolution Framework for Volume Rendering.* ACM Siggraph Symposium on Volume Visualization, 51-57, 1994.
12. T. Totsuka and M. Levoy. *Frequency Domain Volume Rendering.* ACM Siggraph Computer Graphics Proceedings, (Annual Conference Series):271-278, 1993.
13. P. Lacroute and M. Levoy. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation.* ACM Siggraph Computer Graphics Proceedings, (Conference Series):451-458, 1994.
14. C.A. Shaffer. *Bit Interleaving for Quad- or Octrees* In "Graphics Gems I",443-447, Academic Press 1990
15. T. Porter and T. Duff. *Compositing Digital Images.* ACM Computer Graphics, 18(3), 1984.
16. D.P. Anderson. *Hidden Line Elimination in Projected Grid Surfaces.* ACM Transactions on Graphics, 1(4): 274-288, 1982.

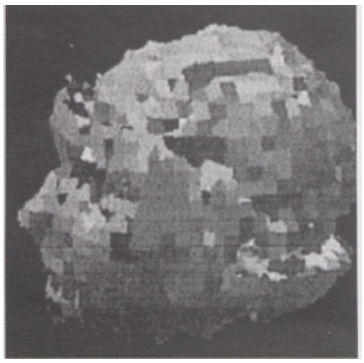

Plate 1 - Parallel rendering of MR Brain volume approximated using the first 64Kb of data.

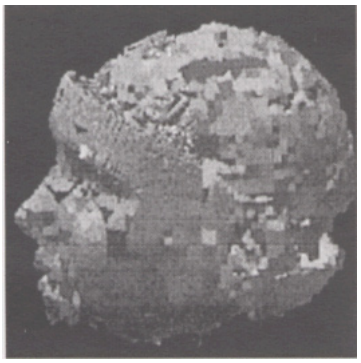

Plate 2 - Parallel rendering of MR Brain volume approximated using the first 256Kb of data. Note that the forehead area is no longer being approximated and the approximation at the rear of the head has improved.

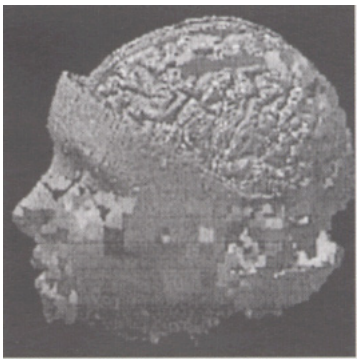

Plate 3 - Parallel rendering of MR Brain volume approximated using the first 512Kb of data. Note that most of the upper areas of the head are no longer being approximated.

Plate 4 - Full parallel octree rendering of the MR Brain volume without filtering. The full data size was 1.8Mb.

Plate 5 - Full RLE rendering of the MR Brain volume with filtering. The full data size was 1.6Mb.

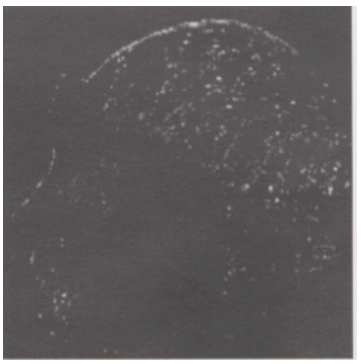

Plate 6 - Subtraction of plates 4 and 5 (Lighter areas show larger differences). The profile lines around the top left of the image are due to the slight correction in the shear which the filtering performs.