UC Davis
IDAV Publications

Title
Efficient Error Calculation for Multiresolution Texture-Based Volume Visualization

Permalink
https://escholarship.org/uc/item/56c4v20q

Authors
LaMar, Eric C.
Hamann, Bernd
Joy, Ken

Publication Date
2003

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Efficient Error Calculation for Multiresolution Texture-based Volume Visualization

Eric LaMar*,
Bernd Hamann, and Kenneth I. Joy**

Center for Applied Scientific Computing (CASC)
Lawrence Livermore National Laboratory
Livermore, CA 94550, USA

Abstract Multiresolution texture-based volume visualization is an excellent technique to enable interactive rendering of massive data sets. Interactive manipulation of a transfer function is necessary for proper exploration of a data set. However, multiresolution techniques require assessing the accuracy of the resulting images, and re-computing the error after each change in a transfer function is very expensive. We extend our existing multiresolution volume visualization method by introducing a method for accelerating error calculations for multiresolution volume approximations. Computing the error for an approximation requires adding individual error terms. One error value must be computed once for each original voxel and its corresponding approximating voxel. For byte data, i.e., data sets where integer function values between 0 and 255 are given, we observe that the set of "error pairs" can be quite large, yet the set of *unique* error pairs is small. Instead of evaluating the error function for each original voxel, we construct a table of the unique combinations and the number of their occurrences. To evaluate the error, we add the products of the error function for each unique error pair and the frequency of each error pair. This approach dramatically reduces the amount of computation time involved and allows us to re-compute the error associated with a new transfer function quickly.

1 Introduction

When rendering images from approximations, it is necessary to know how close a generated image is to the original data. For multiresolution volume visualization, it is not possible to compare the images generated from original data to all possible images generated from approximations. The reason for using the approximations is to substantially reduce the amount of time required to render the data. If we assume that there is a reasonable amount of correlation between the data and the resulting imagery, we can compute an error value between the approximations and the original data (in 3D object space). We can then use that value to estimate the error in the 2D imagery.

Even so, the amount of time required to evaluate the error over an entire data hierarchy can be significant. This consideration is especially important when it is necessary for a user to interactively modify a transfer function: each change in the transfer function requires us to re-compute the error for the entire hierarchy.

We introduce a solution based on the observation that, for many data sets, the range size of scalar data sets is often many orders smaller that the domain size (physical extension) – and that, instead of evaluating an error function for each original voxel, it suffices to count the frequencies of unique pairs of error terms. In the case of 8-bit (byte) integer data, there are only 256^2 combinations (each term is a single byte, or 256 possible values). To compute the error, instead of adding individual error terms, we add the products of the error (for a unique pair of error terms) and the frequency of that unique pair.

For example, a typical 512^3 voxel data set, with one byte per voxel, contains 2^{27} bytes. To compute the error, a naive method would evaluate an error function for each of the original 2^{27} voxels. However, there are only 256^2 *unique* pairs of error terms. Thus, to compute the error, we evaluate the error function for each unique pair of error terms, or 2^{16} times – which is 2^{11} times faster than the naive method. This algorithm requires us to examine the entire data set for unique pairs of error terms - this is a preprocessing cost that can be performed off-line.

* lamar1@llnl.gov, Center for Applied Scientific Computing (CASC), Lawrence Livermore National Laboratory, Box 808, L-661, Livermore, CA 94550 U.S.A.

** {hamann,joy}@cs.ucdavis.edu, Center for Image Processing and Integrated Computing (CIPIC), Dept. of Computer Science, University of California, Davis, CA 95616-8562

This work is a direct extension of earlier work reported in [LJH99], [LHJ00], and [LDHJ00]. We first review the generation and rendering of a volume hierarchy in Section 2, discuss the error criterion we use in Section 3, cover some basic optimizations of the general approach in Section 4, provide performance statistics in Section 5, and discuss directions for future work in Section 6.

2 Multiresolution Volume Visualization

Our multiresolution volume visualization system uses hardware-accelerated 3D texturing for rendering a volume; it represents a volume with index textures. A transfer function is applied by first building a look-up table, transferring it into the graphics system, and then transferring the texture tile. The translation of index values to display values (luminance or color) is performed in hardware. In the following discussion, we use the term *tile* to refer to the data and *node* to refer to the element of a binary tree or octree. In some passages, we will use *tile* to refer to both. *Voxels* are attributes of *tiles* and spatial location and extent are attributes of *nodes.*

2.1 Generating the Hierarchy

First, we review certain aspects of our multiresolution data representation and how it influences the decisions on how to evaluate and store error. The underlying assumption is that data sets are too large to fit into texture memory. Data sets are often too large even to fit into main memory. Given a volumetric data set, we produce a hierarchy of approximations. Each level in the approximation hierarchy is half the size of the next level. Each level is broken into constant-sized tiles – tiles that are small enough to fit in their entirety in texture memory, see [LJH99]. This is also called “bricking” in [GHY98]. Figure 1 shows the decomposition of a block consisting of 29

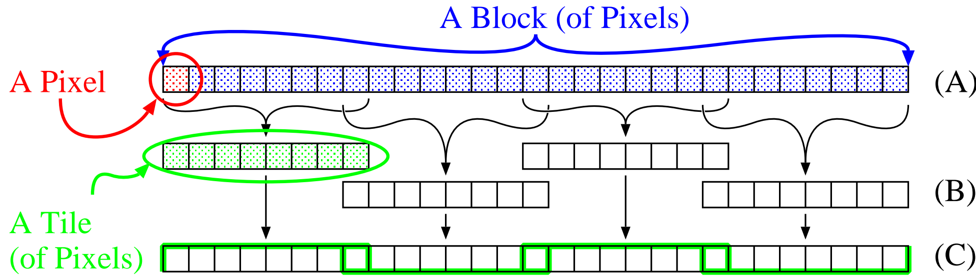


Figure1. Line (A) shows a block of pixels, is broken into four tiles of eight pixels on line (B). Line (C) is our shorthand notation for line (B).

pixels (line A) into four tiles of eight pixels (line B), with one shared pixel in the regions where the tiles overlap. The shared pixel is necessary because we linearly interpolate pixel centers. The alternating thicker borders in line (C) show the position of the individual tiles of a block - this is our “shorthand notation” for line (B).

Figure 2 shows a one-dimensional texture hierarchy of four levels. The top level, level 0, is the original texture (of 57 pixels), broken into eight tiles (of eight pixels). Level 1 contains four tiles at half of the original resolution (29 pixels), and so on. The dashed vertical lines on either side show the domain of the texture function over the hierarchy. Arrows indicate the parent-child relationship of the hierarchy, defining a binary tree, rooted at the coarsest tile, level 3. The bold vertical line denotes a point of interest, p, and tiles are selected when the distance from p to the center of the tile is greater than the width of the tile. One starts with the root tile and performs selection until all tiles meet this distance-to-width criterion, or no smaller tiles exist[1]. The double-headed vertical arrows show selected tiles and their correspondence in the final image.

Figure 3 shows a two-dimensional quadtree example. The original texture, level 0, has 256 tiles. The shaded regions in each level show the portion of that level used to approximate the final image. The selection method in the two-dimensional case is similar to the one for the one-dimensional case: A node is selected when the distance from the center of the node to the point p is greater than the length of the diagonal of the node. The original

[1] This is the case on the left side of Figure 2.

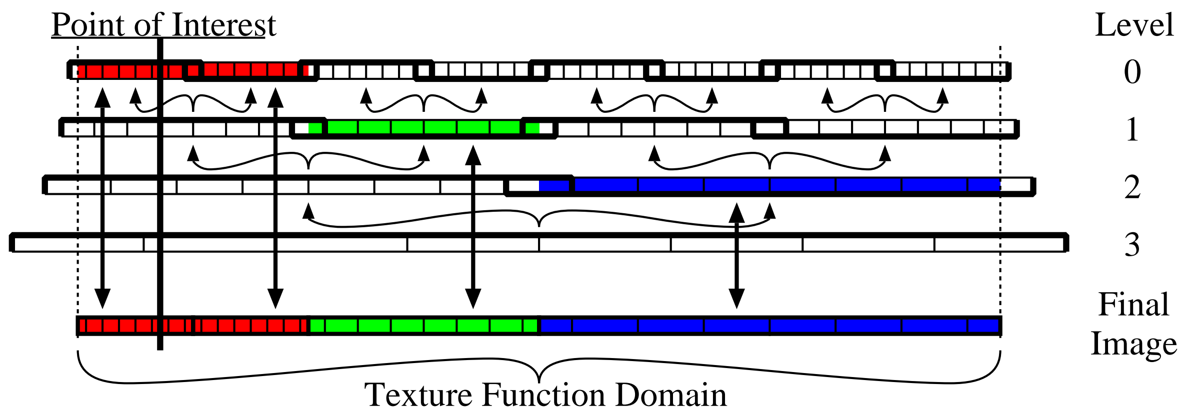


Figure2. Selecting from a texture hierarchy of four levels. Level 0 is the original texture, broken into eight tiles. The dashed lines show the domain of the texture function over the hierarchy. The bold vertical line represents a point p of interest. Tile selection depends on the width of the tile and the distance from the point. The red, green, and blue shaded regions in levels 0, 1, and 2, respectively, and then in the Final Image, show from which levels of detail the data is used to create the final image.

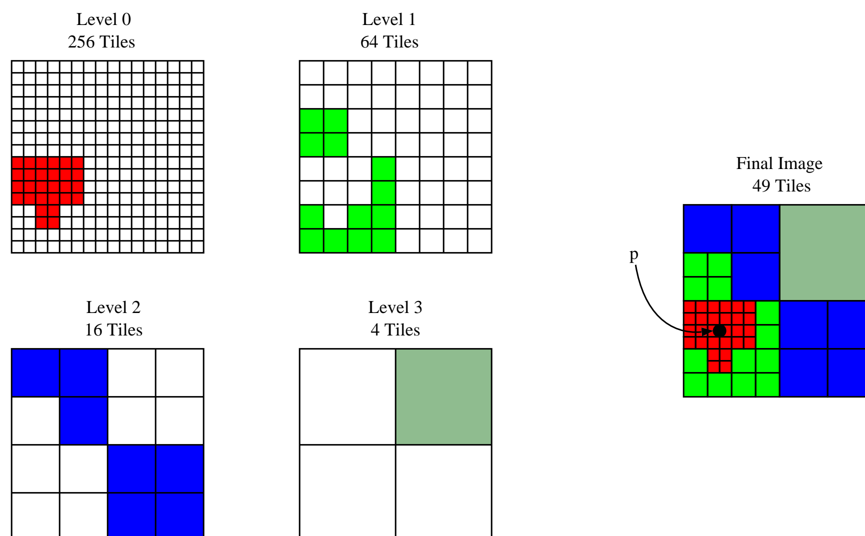


Figure3. Selecting tiles in two dimensions from a texture hierarchy of five levels (Level 4 not shown). Given the point p, tiles are selected when the distance from the center of the tile to p is greater than the length of the diagonal of the tile. Tiles selected from each level are shaded.

texture is divided into 256 tiles. The adaptive rendering scheme uses 49 tiles or, roughly, one-fifth of the data. This scheme can easily be used for three-dimensional textures using an octree.

2.2 Node Selection and Rendering

The first rendering step determines which nodes will be rendered. This means finding a set of non-overlaying nodes that meet some error criterion. The general logic is to subdivide nodes, starting at the root node, until the error criterion is met, always subdividing the node with the greatest error.

We use a priority queue of nodes that is sorted by descending error, *i.e.*, the first node in the queue is the one with the highest error. We first push the root node onto the queue. We then iterate the following: we examine to top node (the node with the highest error). If the error criterion is not met (which we define below), that node is subdivided, or removed from the queue and all of its children are added to the queue. This continues until the error criterion is met. This is guaranteed to terminate as all leaf nodes have zero error. All nodes still in the queue meet the error criterion and are rendered.

Our primary selection filter is based on one of these two error criterion:

- L-infinity (l_∞): Subdivide the node if the node's associated l_∞ error is greater than some maximum value, *i.e.*, all rendered nodes must have an error less than this maximum value.
- Root-mean-square (RMS): Subdivide the node if the root-mean-square error over all selected nodes is greater than some maximum value, *i.e.*, the root of the sum of the squared differences considering all rendered nodes must be less than this maximum value. Rather than re-calculate the error each time a node is subdivided, we keep a running error total, subtracting a node's error when it is removed from the queue and adding a node's error when it is added to the queue.

We note that many other error criterion can be used and that our algorithm is not limited to the two selected here. Nodes are sorted and composited in back-to-front order. We order nodes with respect to the view direction such that, when drawn in this order, no node is drawn behind a rendered node. The order is fixed for the entire tree for orthogonal projections and has to be computed just once for each new rendering [GHY98]. For perspective projections, the order must be computed at each node.

3 Error Calculation

We calculate error on a per-node basis: When a node meets the error criterion, it is rendered; otherwise, its child (higher-resolution) nodes are considered for rendering. We currently assume a piece-wise constant function implied by the set of given voxels, but use trilinear interpolation for the texture. This approach simplifies the error calculation.

We use two error norms: the L-infinity and root-mean-square (RMS) norms. Again, we note that other error norms may be better, but these are sufficiently simple as to not complicate the following discussion. Given two sets of function values, $\{f_i\}$ and $\{g_i\}$, $i = 0, \ldots, n-1$, the L-infinity error norm is defined as $l_\infty = max_i \{|x_i|\}$, and the RMS is defined as $E_{rms} = \sqrt{\frac{1}{n}\sum_i (x_i^2)}$, where $x_i = T[f_i] - T[g_i]$ and $T[x]$ is a transfer function. For the purposes of this discussion, we assume that $T[x]$ is a simple scalar function, mapping density to gray-scale luminance; the issue of error in color space is beyond the scope of this discussion. We evaluate the error function once for each Level-0 voxel.

A data set of size 512^3 contains 2^{27} voxels, with the same number of pairs of error terms for each level of the hierarchy. However, when using byte data, we observe that, though there are 2^{27} pairs of values, there are only 2^{16} (256^2) unique pairs of error values. This means, on average, that each unique pair is evaluated 2^{11} times.

Our solution is to add a two-dimensional table Q to each internal (*e.g.*, approximating) node of the octree, see Figure 5. The elements $Q_{a,b}$ store the numbers of occurrences for each (f_i, g_i) pair, where a and b are the table indices corresponding to f_i and g_i, respectively. Thus

$$Q_{a,b} = \sum_i \begin{pmatrix} 1 \text{ if } f_i = a \text{ and } g_i = b \\ 0 \text{ otherwise} \end{pmatrix}.$$

The tables are created only once, when the data is loaded, and count the number of occurrences of unique error terms that the node "covers" of the original data.

To calculate the L-infinity error for a node, we search for the largest value, with the requirement that this value corresponds to a real pair of values:

$$l_\infty = max_{a,b|Q_{a,b}\neq 0}\left\{|T[a]-T[b]|\right\}.$$

We compute the RMS error for a node as

$$E_{rms} = \sqrt{\frac{1}{n}\sum_{a,b}\left(T[a]-T[b]\right)^2 \times Q_{a,b}},$$

where $n = \sum_{a,b}(Q_{a,b})$. For a set of t nodes, we compute the RMS value

$$E_{rms} = \sqrt{\frac{1}{N}\sum_{t}\left(\sum_{a,b}\left(T[a]-T[b]\right)^2 \times Q_{a,b}\right)},$$

where $N = \sum_t\left(\sum_{a,b} Q_{a,b}\right)$.

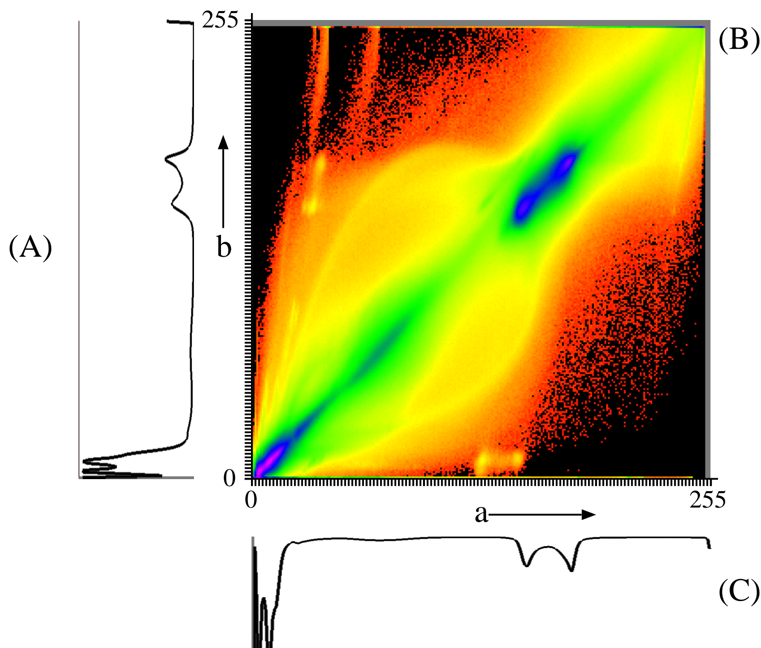


Figure4. The Visible Female CT data set. Image (B) in the upper-right corner shows the frequency relationship of the original and first approximation of the Visible Female CT data set. The image consists of 256 × 256 pixels, with a on the horizontal axis and b on the vertical axis; each pixel corresponds to a $Q_{a,b}$ element. This particular table, covering all of the original domain, would not be produced in practice; normally, Q tables associated with the first-level approximation cover fairly small regions of the original domain. Image (B) is shown here to provide the reader with insight into the nature of a typical Q table. The colors are assigned by normalizing the logarithm of the number of occurrences of a $Q_{a,b}$ element, linearly mapped to a rainbow color sequence, where zero maps to red and one maps to violet: $pixel_{a,b} = RainbowColorMap\left[ln\left(Q_{a,b}\right)/ln\left(Max_{a,b}\left\{Q_{a,b}\right\}\right)\right]$. Graph (A), on the left, shows the histogram of the original data (Level 0), with positive frequency pointing left. Graph (C), on the bottom, shows the histogram of the first approximation (Level 1), with positive frequency pointing down.

Image 4(B), shown in the upper right-hand corner of Figure 4, shows a graphical representation of the frequency relationship of the original and first approximation of the Visible Female CT dataset. Figure 4 indicates that the approximation is generally good: Most of the error terms are along the diagonal. The "lines" in the upper-left

corner of image 4(B) may correspond to high gradients present in the data set. (The Visible Female CT data set was produced such that sections of the body fill a 512^2 image: these different sections are scanned with different spatial scales. We have not accounted for these different spatial scales in our version of the data set, thus there exist several significant discontinuities in data values in the data set.)

4 Optimizations

Evaluating a table is still fairly expensive when interactive performance is required. We currently use three methods to reduce the time needed to evaluate the error.

First, one observes that the order of the error terms for calculating L-infinity and root-mean-square errors does not matter: $x_i = |f_i - g_i| = |g_i - f_i|$ and $x_i = (f_i - g_i)^2 = (g_i - f_i)^2$. If we order the indices in $Q_{a,b}$ such that $a \leq b$, or $a = min(f_i, g_i)$ and $b = max(f_i, g_i)$, we obtain a triangular matrix that has slightly more than half the terms of a full matrix (32896 vs. 65536 entries). This consideration allows us to roughly halve the evaluation time and storage requirements.

Second, one observes that, typically, there is a strong degree of correlation between approximation and original data. This means that the values of $Q_{a,b}$ are large when a is close to b (i.e., near the diagonal); and small, often zero, when $a \ll b$, (i.e., far away from the diagonal). We have observed that the number of non-zero entries in a Q table decreases (i.e., the table is becoming more sparse) for nodes closer to the leaves of the octree. This happens since the nodes closer to the leaf nodes correspond to high-resolution approximations and thus a better approximation - the correlation is strong, and the non-zero values in the Q table cluster close to the diagonal. Also, the nodes closer to the leaves cover a progressively smaller section of the domain. We perform a column-major scan, i.e., traverse the table first by column, then by row. Thus, the Q tables become sparse, and we can terminate the checking of elements if we remember the last non-zero entry for a row. We maintain a table, L_{row}, that contains the index of the last non-zero value for a row. It may even be possible to perform run-length encoding for a row to skip over regions of zero entries. Figure 4 shows that there are large, interior regions with zero entries.

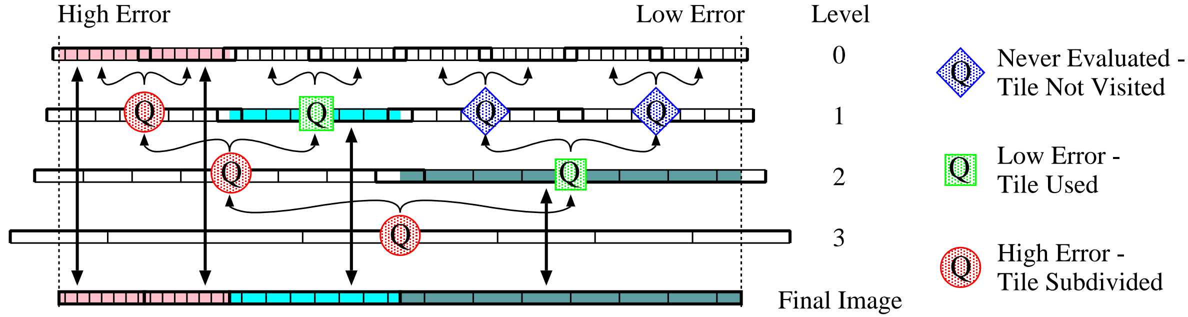
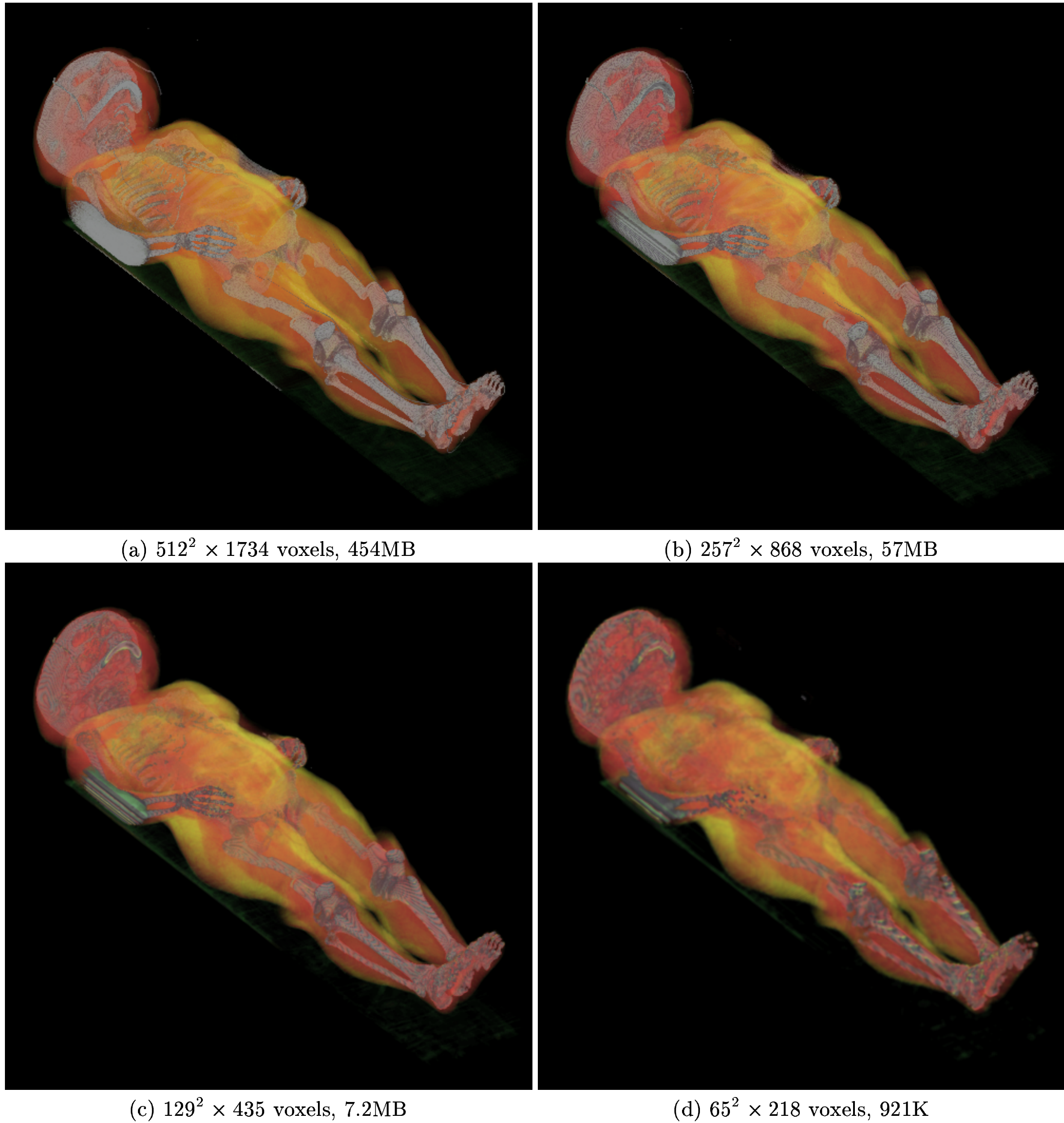


Figure5. Selecting from a texture hierarchy consisting of four levels with lazy evaluation of error. The error selection criterion selects high-resolution nodes in the high-error regions (left) and low resolution nodes in the low-error regions (right). The letter Q indicates nodes with a Q table, i.e., nodes that are all internal (approximating) nodes. Three classifications of tiles are shown. Blue-diamond tiles are never visited. Error is evaluated in red-round and green-square tiles; red-round tiles exceed the error requirement and are subdivided; green-square tiles meet the error requirement and are used.

Third, one can use “lazy evaluation” of the error. When we re-calculate the error for all nodes in a hierarchy, and few of the nodes are rendered, much of the error evaluation is done but not used. Thus, for each new transfer function, we only re-calculate the error value of those nodes that are being considered for rendering, see [LHJ00]. In Figure 5, nodes with a red circle around the Q do not satisfy the error criterion, nodes with green squares around the Q meet the error criterion, and nodes with blue diamonds around the Q are never visited, and thus no error calculation is performed.

5 Results

Performance results hierarchy generation, error table generation, and error computation time were obtained on an SGI Origin2000 with 10GB of memory, using one (of 16) 195MHz R10K processors. However, images were produced on an SGI Onyx2 Infinite Reality with 512MB of memory, using one (of four) 195MHz R10K processors. This was done for the following reasons: We are interested in the time required to process the hierarchy, free of memory limitations. The logical memory used during a visualization was 2.4GB, and thrashing completely dominated (by a factor of 10 or more) hierarchy/error-table generation and error calculation times when performed on the Onyx2. The SGI Origin2000 does not have a graphics subsystem, while the SGI Onyx2 does.



(a) $512^2 \times 1734$ voxels, 454MB

(b) $257^2 \times 868$ voxels, 57MB

(c) $129^2 \times 435$ voxels, 7.2MB

(d) $65^2 \times 218$ voxels, 921K

Figure6. Visible Female CT data set rendered at four resolutions (see Table 1). The transfer function shows bones in white, fat in yellow, muscle in red, and internal organs in green. Different spatial scales were used for different sections of the body during data acquisition.

Image	Level	Voxels	Nodes in Level	Nodes Rendered	Mem. (MB)	Time (sec.)	Error l_∞	E_{rms}
6a	0	$512^2 \times 1734$	2268	1560	390	83.4	0.000	0.00000
6b	1	$257^2 \times 868$	350	263	65	8.73	0.305	0.00678
6c	2	$129^2 \times 435$	63	49	13	2.38	0.305	0.00917
6d	3	$65^2 \times 218$	16	16	4	0.563	0.305	0.01059

Table1. Visible Female CT data set statistics.

We have used the Visible Female CT data set, consisting of $512^2 \times 1734$ voxels in our experiments. Figure 6 shows different-resolution images of this data set: Image 6(a) shows the original data, and images 6(b) to 6(d) are progressively 1/2 linear (1/8 total) size. Table 1 summarizes performance statistics for four (of six) levels of the hierarchy. The "Voxels" column shows the total size of the approximation in voxels, and the next column shows the total number of nodes associated with that level. The "Nodes Rendered" column shows how many nodes where used to render that level. The number of nodes rendered is actually less than one would expect: Many regions of the data set are constant, so there is no error when approximating these regions. Each tile contains 64^3 bytes = 256K. When transferring n tiles, the total memory transferred to the graphics subsystem for n tiles is 256K$\times n$, shown in the "Memory" column. The "Time" column lists the times, in seconds, required to produce the rendering for the various levels. The l_∞ and E_{rms} columns show the error values associated with those renderings.

Step	Index Texture
Compute Texture Hierarchy	1 min 46 sec
Compute Error Table	5 min 35 sec
Calculate Error	1.23 sec

Table2. Performance statistics for Visible Female CT data set.

Table 2 summarizes the times for various stages of our system. We note that the "Compute-Texture-Hierarchy" and "Compute-Error-Tables" stages are only performed once for a data set, and only the "Calculate-Error" step is performed for each new transfer function. The time for "Calculate-Error" is the times needed to re-compute the error for all 432 internal (i.e., approximating) nodes. The time per node is approximately 0.0028 seconds – and when coupled with a lazy evaluation scheme, "Calculate-Error" time is very small relative to the other parts of the rendering pipeline (see "Time" column in Table 1).

6 Conclusions and Future Work

We believe that there are several directions to continue this work. The first directions is to extend this technique to color images or more general vector-valued data hierarchies. The second direction is to consider data other than byte data: data sets with 12 and 16 bits per voxel are also common. However, a table would contain 4096^2 (2^{24}) or 65536^2 (2^{32}) entries. These tables would be larger than the actual volume data per node, and possibly require more time for evaluation. Could some quantizing approach work? Since these nodes should have a high degree of correlation, will these tables be sufficiently sparse to compress? A third direction is to apply our technique to time-varying data. The error between nodes in different time steps would be expressed in the same manner and could be encoded in a table.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. This work was supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract

N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; the Lawrence Berkeley National Laboratory; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, and the members of the Data Analysis and Exploration thrust of the Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory.

References

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In Arie Kaufman and Wolfgang Krueger, editors, *1994 Symposium on Volume Visualization*, pages 91–98. ACM SIGGRAPH, October 1994.

[GHY98] Robert Grzeszczuk, Chris Henn, and Roni Yagel. *SIGGRAPH '98 "Advanced Geometric Techniques for Ray Casting Volumes" course notes.* ACM, July 1998.

[LDHJ00] Eric C. LaMar, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Texturing-based Rendering of Arbitrarily Oriented Cutting-Planes. In W. C. de Leeuw and R. van Liere, editors, *Data Visualization 2000*, pages 105–114, Vienna, Austria, May 29-30, 2000. Proceedings of VisSym '00 – The Joint Eurographics and IEEE TVCG Conference on Visualization, Springer-Verlag.

[Lev87] Marc Levoy. Display of Surfaces from Volume Data. *Computer Graphics and Applications*, 8(3):29–37, February 1987.

[LHJ00] Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Hardware Texturing-based Volume Visualization. In R. F. Erbacher, P. C. Chen, J. C. Roberts, and Craig M. Wittenbrink, editors, *Visual Data Exploration and Analysis*, pages 365–374, Bellingham, Washington, January 2000. SPIE – The International Society for Optical Engineering.

[LJH99] Eric C. LaMar, Kenneth I. Joy, and Bernd Hamann. Multi-Resolution techniques for Interactive Hardware Texturing-based Volume Visualization. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization 99*, pages 355–361. IEEE, ACM Press, October 25-29, 1999.

[WE98] Rüdiger Westermann and Thomas Ertl. Efficiently Using Graphics Hardware In Volume Rendering Applications. In Shiela Hoffmeyer, editor, *Proceedings of Siggraph 98*, pages 169–177. ACM, July 19-24, 1998.