Multi-Dimensional Trees for Controlled Volume Rendering and Compression*

Jane Wilhelms and Allen Van Gelder

UCSC-CRL-94-02

Baskin Center for Computer Engineering and Information Sciences
University of California, Santa Cruz 95064
wilhelms@cs.ucsc.edu avg@cs.ucsc.edu

Jan. 21, 1994 (rev.)

Abstract

This paper explores the use of multi-dimensional trees to provide spatial and temporal efficiencies in imaging large data sets. Each node of the tree contains a model of the data in terms of a fixed number of basis functions, a measure of the error in that model, and a measure of the importance of the data in the region covered by the node. A divide-and-conquer algorithm permits efficient computation of these quantities at all nodes of the tree. The flexible design permits various sets of basis functions, error criteria, and importance criteria to be implemented easily.

Selective traversal of the tree provides images in acceptable time, by drawing nodes that cover a large volume as single objects when the approximation error and/or importance are low, and descending to finer detail otherwise. Trees over very large datasets can be pruned by the same criterion to provide data representations of acceptable size and accuracy. Compression and traversal are controlled by a user-defined combination of modeling error and data importance. For imaging decisions additional parameters are considered, including grid location, allowed time, and projected screen area. To analyse results, two evaluation metrics are used: the first compares the hierarchical model to actual data values, and the second compares the pixel values of images produced by different parameter settings.

*Draft; earlier version was submitted to *SIGGRAPH* 1994; subject to revision.

1 Introduction

As computers and algorithms improve so do our expectations of the kind and quality of images that can be produced. In scientific visualization, many data sets are larger than can be visualized in a comfortable amount of time, or even can be read into the available memory. The research described here explores the use of multi-dimensional trees to deal with both the spatial and temporal aspects of this problem.

Our particular problem area is visualization of sampled k-dimensional scalar data arranged on a regular grid. Our visualization method is direct volume rendering. However, we believe the data representation paradigm we use is applicable to more general multivariate and non-rectilinear data sets, and also can provide useful insights into the imaging of any large graphical database.

Our approach is to build a space-efficient hierarchy over the data, each node of which contains three types of information: a model of the data below it; error and evaluation information for selective traversal; and structural information. The user defines acceptable tolerances for evaluation parameters, and selective traversal of the tree defines that part of the hierarchy within those tolerances. Nodes beneath this selected subset of the tree can be pruned, resulting in an alternate, often more succinct, representation of the data. For imaging, the shallowest regions of the selected tree that lie within the tolerances are drawn.

We have found that selective traversal produces images that are both subjectively and quantifiably very close to those produced using the entire data set, but is significantly faster. It provides an extremely flexible tool for creating error-controlled images in acceptable time. Furthermore, by storing the selected portion of the tree, the method can also provide data compression.

2 Background and Related Work

Work most closely related to ours is that concerned with hierarchical data structures for controlled imaging, algorithms for fast volume rendering, and methods for dealing with large data sets.

Meagher did some of the earliest work in representing 3D data using octrees [Mea82], and many variations have appeared over the years. Levoy used a binary octree to avoid regions whose data was transparent [Lev89, Lev90]. Wilhelms and Van Gelder used a max-min octree to avoid regions not intersecting the desired isosurface, and presented a space-efficient subdivision strategy, called *branch on need* (BON) [WVG92]. This paper extends octrees and the BON strategy to k dimensions.

Laur and Hanrahan build an octree over voxels, and compute the data mean and root mean square error (RME2) at each node [LH91]. This permits volume rendering by progressive refinement, with the user specifying an error tolerance. Nodes with RME2 within the tolerance are rendered as single "splats". Our work builds upon that paper, and extends it in several ways:

1. Data models other than a constant (the mean) are supported, and computed throughout the tree in constant time per node (Section 3.3). In particular, a trilinear model has been implemented. Other polynomial models are easily incorporated, and the design allows for other sets of orthogonal basis functions.

2. Both voxel and cell conventions are supported.

3. Error metrics based on L_q norms are supported (Section 4.3.1); RME2 corresponds to the L_2 norm. Experiments indicate that higher values of q give better images for the same number of rendered objects (Section 8.2).

4. Errors can be weighted by an "importance" function (Section 5).

5. We have quantified image differences (Section 8.2).

Funkhauser and Sequin used a hierarchy for gaining consistent frame rate for complex viewing environment [FS93]. They also used a weighted combination of parameters to control imaging.

Other than using hierarchies, speed gains for direct volume rendering have been achieved by using voxel splatting [Wes89, Wes90], hardware-assisted projection [ST90, LH91, WVG91], and preprocessing [Cha93, DFM87, Wil92, VGW93]. Preprocessing, however, often involves creating large auxiliary data structures, which we are particularly trying to avoid in the research presented here.

Our method of hierarchical data representation has some similarities to wavelets and multi-resolution analysis [Mal89, Chu92, Mur93, GSCH93], but it has several significant differences:

1. When a region is divided in two, the two subdivisions are not necessarily of the same length. Lengths that are not powers of two are handled naturally.

2. Basis functions on the same level, or "scale", are orthogonal, but those on different levels may not be. Basis functions are not normalized.

3. There is no dilation, and no single "mother wavelet". Several independent basis functions may exist on the same level in the same interval, such as a constant, a linear, and a quadratic function.

4. "Detail" functions are not used. At any point, the value of the model function depends on basis functions at a single level only, not on a sum over all levels.

5. Basis functions at the same level have either identical supports, or disjoint supports; there is no overlapping.

For example, Muraki used multi-resolution analysis to represent 3D volumes [Mur93]. In constrast to our method, basis functions overlapped, and the calculation of one function value involved as many as 2000 basis functions.

Malzbender described efficient volume rendering through the use of Fourier transforms [Mal93]. Levoy described a variation that included a lighting model [Lev92]. Neither method can model opacity.

Ning and Hesselink [NH92, NH93] used vector quantization to produce compressed data sets that could be rendered directly. While this approach gives very good compression, it is not as flexible as a hierarchical model for imaging.

3 Hierarchical Data Models

This section describes the techniques to compute data models and approximation errors at all nodes of a multi-dimensional tree. Efficiency is achieved by computing the model and error at each node in terms of those values for the node's children. The set of basis functions for the model is fixed for a given tree, but there is considerable flexibility in choosing this set.

3.1 Notation

We will be using notation for k-dimensional space of reals, $\mathbf{R}^k$. In practice, k is usually 3 or 4. In general, bold face letters represent k-D vectors. Thus, *location* in k-D space is denoted by $\mathbf{x} = (x_1, \ldots, x_k)$. In 3-D and 4-D we will often use (x, y, z) and (x, y, z, t).

A *volume* in $\mathbf{R}^k$ is a rectangular k-D parallelopiped, or closed *interval* denoted as $[\mathbf{x}_{min}, \mathbf{x}_{max}]$. That is, point $\mathbf{x}$ is in the volume if and only if $x_{min,j} \leq x_j \leq x_{max,j}$ for $j = 1, \ldots, k$. The *width* in dimension j is denoted by $w_j = x_{max,j} - x_{min,j}$; the vector of widths is $\mathbf{w}$.

The *volumetric data* is given as discrete samples on a regular k-D grid of resolutions $\mathbf{r} = (r_1, \ldots, r_k)$. Sample data points are indexed by a k-D index $\mathbf{p}$, where p_j runs from 1 to r_j. The *spacings* of the data are

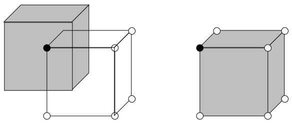

Figure 1: A *voxel* is the region surrounding a data point, whereas a *cell* is the region between data points.

$\mathbf{\Delta x}$. The relationship of the grid to $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ depends on whether we are using the *voxel* convention, or the *cell* convention (see Figure 1).

1. In the *voxel* convention, $w_j = r_j \Delta x_j$, $x_{min,j} = -\frac{1}{2} r_j \Delta x_j$, and $x_{max,j} = \frac{1}{2} r_j \Delta x_j$. Each voxel is considered to have a sample data point at its centroid. Thus, the location associated with index $\mathbf{p}$ is $((p_1 - \frac{1}{2} r_1)\Delta x_1, \ldots, (p_k - \frac{1}{2} r_k)\Delta x_k)$.

2. In the *cell* convention, $w_j = (r_j - 1)\Delta x_j$, $x_{min,j} = -\frac{1}{2}(r_j - 1)\Delta x_j$, and $x_{max,j} = \frac{1}{2}(r_j - 1)\Delta x_j$. Each cell is considered to have sample data points at each of its corners. Thus, the location associated with index $\mathbf{p}$ is $((p_1 - \frac{1}{2} r_1 + \frac{1}{2})\Delta x_1, \ldots, (p_k - \frac{1}{2} r_k + \frac{1}{2})\Delta x_k)$.

Values of the sample data are denoted by $g(\mathbf{p})$, and the data viewed as a function throughout the volume is $g(\mathbf{x})$.

3.2 Inner Products and Orthogonality

The derivation and error analysis of the hierarchical representation rest upon the concept of inner product. Indeed, by conducting the analysis in terms of a general inner product, many of the complicated details of particular choices are avoided, and we retain the flexibility to use different inner products with only minor changes to the implementation. Typical inner products of interest are integrals over the volume and sums over the grid points. Integrals and sums may be weighted or unweighted.

For two functions f and g, defined on R^k, and belonging to a suitable function space, let $\langle f, g\rangle$ denote their inner product. (We are not interested in the most general possible function space; let us take it to be functions with piecewise-continuous second derivatives on the k-D interval $(\mathbf{x}_{min}, \mathbf{x}_{max})$.) Recall that an inner product is any operation satisfying the properties, or axioms:

1. $\langle f, g\rangle = \langle g, f\rangle$ and $\langle f, g+h\rangle = \langle f, g\rangle + \langle f, h\rangle$.
2. For any scalar c, $\langle f, cg\rangle = c\langle f, g\rangle$
3. $\langle f, f\rangle \geq 0$, with equality if and only if $f = 0$.

An inner product on a volume induces inner products on subvolumes by restriction, but care must be taken to weight the points on the boundaries of several subvolumes so that their total weight is the same as in the original volume. This problem arises in practice only for a cell-based volume and a sum-based inner product. Our implementation avoids the complications by using an integral-based inner product.

Two functions f and g are *orthogonal* if $\langle f, g\rangle = 0$. An inner product induces a *norm*:

$$||f|| = \sqrt{\langle f, f\rangle}$$

and the *distance* between f and g can be defined as $||g - f||$.

If function f is *separable* as $f(x, y) = f_1(x) f_2(y)$, then $\langle f, f\rangle = \langle f_1, f_1\rangle\langle f_2, f_2\rangle$. The relationship extends to any number of variables.

Now suppose we have a set $\mathcal{B}$ of *basis functions*, $\{b_i\}$, such that distinct functions in $\mathcal{B}$ are orthogonal w.r.t $\langle\rangle$. (We do not assume that $\mathcal{B}$ forms a complete basis for the function space; in fact, we will be interested only in finite $\mathcal{B}$'s.)

For the purpose of approximating g, let us require f to be a weighted sum of basis functions. That is, $f = \sum_i a_i b_i$, where the a_i's are real numbers. Then, as is well known from Fourier theory, f is an *optimal* approximation, in the sense that $\|g - f\|$ is minimized, if and only if

$$\langle g - f, b_i \rangle = 0 \qquad \text{for all } b_i \in \mathcal{B}$$

For optimal f, several important properties will be used in the development:

1. $\langle g - f, f \rangle = 0$.
2. $\langle g - f, g - f \rangle = \langle g, g \rangle - \langle f, f \rangle$.
3. The coefficients of f are given by: $a_i = \dfrac{\langle g, b_i \rangle}{\langle b_i, b_i \rangle}$.
4. $\langle f, f \rangle = \sum_i a_i^2$.

3.3 Divide-and-Conquer Approximation

We are interested in deriving an optimal approximation f to a given function g and *error bounds* (w.r.t. an inner product $\langle\rangle$), over the rectilinear k-D volume V. We seek an expression for f in terms of the optimal approximation and error bounds for each of V_L and V_R. Here, V_L and V_R are a *Left* subvolume and a *Right* subvolume, respectively, that partition V in dimension j. I.e., V is partitioned by a hyperplane orthogonal to the x_j axis.

By using the divide-and-conquer approach, we will be able to compute the optimal coefficients and errors of approximation for all nodes in the tree in constant time per node, and with only one pass through the data.[1]

Let V be the closed k-D interval $[-\frac{1}{2}\mathbf{w}, \frac{1}{2}\mathbf{w}]$. Let the width $w_j = w_L + w_R$, where $w_L > 0$ and $w_R > 0$. Define

$$x_{div,j} = w_L - \tfrac{1}{2}w_j = \tfrac{1}{2}w_j - w_R = \tfrac{1}{2}(w_L - w_R)$$

Let V_L, V_R be the k-D intervals $[(\mathbf{x}_{min})_L, (\mathbf{x}_{max})_L]$ and $[(\mathbf{x}_{min})_R, (\mathbf{x}_{max})_R]$, respectively. Then, $(\mathbf{x}_{min})_L = -\frac{1}{2}\mathbf{w}$ and $(\mathbf{x}_{max})_L = \frac{1}{2}\mathbf{w}$ except that $(x_{max,j})_L = x_{div,j}$. Similarly, $(\mathbf{x}_{min})_R = -\frac{1}{2}\mathbf{w}$ except that $(x_{min,j})_R = x_{div,j}$, and $(\mathbf{x}_{max})_R = \frac{1}{2}\mathbf{w}$. In dimension j, the center of V_L is at $-w_R/2$, and the center of V_R is at $+w_L/2$. In other dimensions, their centers remain at 0.

The desired approximation f over V will use the basis set $\mathcal{B}$. The approximations over V_L and V_R are in basis sets $\mathcal{B}_L$ and $\mathcal{B}_R$, which we assume are closely related to $\mathcal{B}$, but apply to their respective domains. Specifically, we assume that any function in $\mathcal{B}_L$ or $\mathcal{B}_R$ can be obtained as a linear combination of functions in $\mathcal{B}$, restricted to V_L or V_R. This assumption holds for polynomials.

Let $b^{(L)}$ denote basis function $b \in \mathcal{B}_L$, expressed in the coordinate system of V. Also, denote the restriction of a function b to a subvolume V_L by $b|_L$; that is, $b|_L$ is equal to b in V_L and is zero outside V_L. Use similar notations for R.

Example 3.1: Consider a 2-D "volume" V with $\mathbf{w} = (11, 4)$ and $j = 1$ (see Figure 2). Let the set of basis functions be $\mathcal{B} = \{1, x, y, xy\}$. Suppose the desired partition is $w_L = 8$ and $w_R = 3$. The centroid of V_L in the coordinate system of V is at $(-1.5, 0)$.

[1] The computation is also more accurate numerically than summing in a "for loop" through the data.

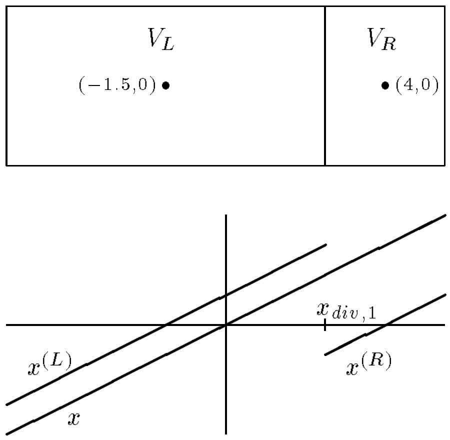

Figure 2: Basis function x for V, V_L and V_R in Example 3.1.

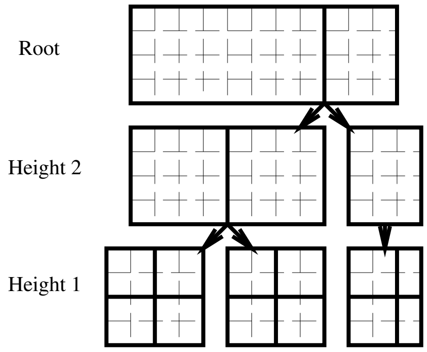

Figure 3: The BON strategy on an 11x4 2D "volume". Solid lines show how nodes subdivide. Dashed lines are voxel or cell boundaries.

The basis set $\mathcal{B}_L$ also has $\{1, x, y, xy\}$, but they are defined in the coordinate system of V_L. Then we have:

$$\begin{array}{rclcrcl} 1^{(L)} &=& 1|_L & \quad & 1^{(R)} &=& 1|_R \\ x^{(L)} &=& x|_L + 1.5|_L & & x^{(R)} &=& x|_R - 4|_R \\ y^{(L)} &=& y|_L & & y^{(R)} &=& y|_R \\ xy^{(L)} &=& xy|_L + 1.5y|_L & & xy^{(R)} &=& xy|_R - 4y|_R \end{array}$$

Observe that the set of linear equations can be solved quickly for $\{1|_L, x|_L, y|_L, xy|_L\}$ in terms of $\{1^{(L)}, x^{(L)}, y^{(L)}, xy^{(L)}\}$.

The idea extends to any number of dimensions: additional dimensions remain centered on 0, like y. Also, higher degree polynomials can be translated, and the translations can be inverted efficiently, as the matrices involved are triangular. $\square$

Because basis functions are not normalized, it is convenient to define *unscaled coefficients*, A_i, by

$$A_i = \langle g, b_i \rangle$$

Then the optimal approximation is $f = \sum_i \left(\frac{A_i}{\langle b_i, b_i \rangle} \right) b_i$. To find A_i, decompose g into $g|_L + g|_R$, the restrictions to V_L and V_R.

$$A_i = \langle g|_L, b_i \rangle + \langle g|_R, b_i \rangle = \langle g|_L, b_i|_L \rangle + \langle g|_R, b_i|_R \rangle$$

But $b_i|_L$ is a linear combination of certain $b_n^{(L)}$, and $b_i|_R$ is a linear combination of certain $b_n^{(R)}$.

We will recursively derive optimal approximations f_L to $g|_L$ and f_R to $g|_R$. Therefore,

$$(A_L)_n = \langle g|_L, b_n^{(L)} \rangle \quad \text{and} \quad (A_R)_n = \langle g|_R, b_n^{(R)} \rangle$$

are known when the two subproblems are completed. These values can be used to compute the A_i. The general description above will now be illustrated for some common basis sets.

Example 3.2: Suppose the basis set $\mathcal{B}$ consists of just the constant function, 1. There is only one unscaled coefficient, A_0, so its subscript can be omitted. We have $A = A_L + A_R$. The scaled coefficient is the (possibly weighted) mean value of g, and is given by

$$a = \frac{\langle 1|_L, 1|_L\rangle a_L + \langle 1|_R, 1|_R\rangle a_R}{\langle 1, 1\rangle}$$

This case is essentially the one considered by Laur and Hanrahan, using the voxel model [LH91]. □

Example 3.3: Suppose $k = 3$ or 4 and $\mathcal{B}$ is the trilinear or quadlinear basis. We shall index $b_i \in \mathcal{B}$ as follows:

$$\begin{array}{rcll} b_0(\mathbf{x}) & = & 1 & \text{for } \mathbf{x} \in V \\ b_1(\mathbf{x}) & = & x_1 b_0(\mathbf{x}) & \\ b_{i+2}(\mathbf{x}) & = & x_2 b_i(\mathbf{x}) & \text{for } i = 0, 1 \\ b_{i+4}(\mathbf{x}) & = & x_3 b_i(\mathbf{x}) & \text{for } i = 0, \ldots, 3 \\ b_{i+8}(\mathbf{x}) & = & x_4 b_i(\mathbf{x}) & \text{for } i = 0, \ldots, 7 \end{array}$$

Thus the bits of the index indicate whether the corresponding linear factor is present, and b_i is independent of x_j just when $\lfloor i/2^{j-1} \rfloor$ is even. We identify (x_1, x_2, x_3, x_4) with (x, y, z, t).

Similarly to Example 3.2, for decomposition in direction j, we have

$$A_i = (A_L)_i + (A_R)_i \qquad \text{for } \lfloor i/2^{j-1} \rfloor \text{ even } (b_i \text{ independent of } x_j)$$

For $\lfloor i/2^{j-1} \rfloor$ odd, we note (see Figure 2) that

$$b_i|_L = b_i^{(L)} - \tfrac{1}{2} w_R b_{i-2^{j-1}}^{(L)} \qquad b_i|_R = b_i^{(R)} + \tfrac{1}{2} w_L b_{i-2^{j-1}}^{(R)}$$

From this it follows that

$$A_i = (A_L)_i + (A_R)_i - \tfrac{1}{2} w_R (A_L)_{i-2^{j-1}} + \tfrac{1}{2} w_L (A_R)_{i-2^{j-1}} \qquad \text{for } \lfloor i/2^{j-1} \rfloor \text{ odd } (b_i \text{ linear in } x_j)$$

□

Example 3.4: Continuing with Example 3.1, assume the indexing is as described in Example 3.3: $b_0 = 1$, $b_1 = x$, $b_2 = y$, $b_3 = xy$. Here $j = 1$, so b_i is linear in x when i is odd. Then we find that

$$\begin{array}{l} A_0 = (A_L)_0 + (A_R)_0 \\ A_2 = (A_L)_2 + (A_R)_2 \\ A_1 = (A_L)_1 + (A_R)_1 - 1.5(A_L)_0 + 4(A_R)_0 \\ A_3 = (A_L)_3 + (A_R)_3 - 1.5(A_L)_2 + 4(A_R)_2 \end{array}$$

□

Example 3.5: Quadratic basis functions can be added to a set that already includes linear and constant functions. Say $b_0 = 1$ and $b_1 = x$. The general form of b_2 on an interval $[-\frac{1}{2}w, \frac{1}{2}w]$ is

$$b_2(x, w) = x^2 - \tfrac{1}{12} w^2$$

We have $\langle b_2, b_2 \rangle = w^5/180$ for the squared-integral inner product.

Now when V is decomposed into V_L and V_R, we get

$$\begin{array}{rcl} b_2(x, w)|_L & = & b_2^{(L)}(x + \frac{1}{2}w_R, w_L) - w_R b_1^{(L)}(x + \frac{1}{2}w_R, w_L) + \frac{1}{6} w_R (w_R - w_L) b_0^{(L)} \\ b_2(x, w)|_R & = & b_2^{(R)}(x - \frac{1}{2}w_L, w_R) + w_L b_1^{(R)}(x - \frac{1}{2}w_L, w_R) + \frac{1}{6} w_L (w_L - w_R) b_0^{(R)} \end{array}$$

The defining equation for A_2 is

$$A_2 = \langle g, b_2 \rangle = \langle g|_L, b_2|_L \rangle + \langle g|_R, b_2|_R \rangle$$

It follows that the recursion equation for A_2 is

$$A_2 = (A_L)_2 + (A_R)_2 - w_R(A_L)_1 + w_L(A_R)_1 + \tfrac{1}{6}(w_L - w_R)\,(w_L(A_R)_0 - w_R(A_L)_0)$$

$\square$

Now the error in the approximation, $e^2 \stackrel{\mathrm{def}}{=} \langle g - f, g - f \rangle$, over the volume V, is computable as follows:

$$\begin{aligned}
\langle g, g \rangle &= \langle g|_L, g|_L \rangle + \langle g|_R, g|_R \rangle \\
\langle f, f \rangle &= \textstyle\sum_i A_i^2 / \langle b_i, b_i \rangle \\
e^2 &= \langle g, g \rangle - \langle f, f \rangle
\end{aligned}$$

To summarize, in each subproblem, V_L and V_R, we compute $\{A_i\}$ and $\langle g, g \rangle$, then use those results to compute these quantities for V. To decompose in several j directions, we decompose in one after the other, recursively. At the bottom level, these quantities are computed directly from the data.

4 Implementation of the Hierarchy

Our hierarchical method uses a flexible k-dimensional tree that encodes structural information about the tree, model information describing the data within the region, and evaluation information to control compression and image quality.

4.1 Structural Information

The hierarchy design is an extension to higher dimensions of the BON octree strategy, described elsewhere [WVG92]. As shown there, this strategy can achieve significant savings when the grid resolutions are unequal, or are not powers of 2; higher dimensions tend to create greater savings. Figure 3 illustrates the main idea in 2D. Our implementation handles up to 8 dimensions, but in practice, we have only used 3- and 4-dimensional data sets.

Two types of structural information are stored within the tree. The first is a pointer to the first child of this node. Sibling tree nodes are contiguous, so one pointer suffices for all. Nodes on height 1 point to the first child in the data. If the data has been compressed (see below), the siblings are again contiguous; otherwise other siblings are locatable in the original grid. If the children of a node are discarded due to compression, that node's child pointer is set to zero. The second type of structural information is the branching pattern, stored as a bit vector.

4.2 Model Information

The model information within each node represents the data beneath that node, either exactly or approximately. In general, the approximation is closest near the bottom of the tree, and gets worse higher up. We have experimented with three data models described below. Only the mean was implemented for arbitrary dimensions; at this point, only 3D data has been used with the two trilinear models.

4.2.1 The Voxel Mean Model

The mean model is a simple voxel model. Each node stores one value representing the average of the data values of all points beneath it. While succinct, the model may show discontinuities in imaging where these regions meet, even when drawn at the deepest voxel level.[2] The spatial cost of storing the mean is one floating point value. For 3D data, there are about $n/7$ nodes for n data points; for 4D, the figure is about $n/15$. The model is easily compressed by truncating the tree and discarding data in its region where desired.

Incidentally, the mean model could be represented with Haar wavelets [Chu92], and, indeed, we first implemented the model in that form. The advantage of the Haar wavelet method is that the tree becomes a complete representation of the data, which is discarded. Disadvantages are that detail functions (7 coefficients in 3D, 15 in 4D) need to be combined to recover the means during tree traversal. As the cost of explicitly storing the mean is only 1/7 the size of the data, we opted for simplicity and did so.

4.2.2 The Voxel Trilinear Model

The voxel trilinear model stores a trilinear function at each node that best fits the data values represented by the node. The trilinear on height one of the tree fits the data points exactly, so the data can be discarded and regenerated as needed from the trilinear coefficients. This model uses the same amount of space as the mean model for data and model combined ($8n/7$ in 3D, $16n/15$ in 4D), and sometimes provides a better approximation of the data at higher levels of the tree.

4.2.3 The Cell Trilinear Model

In this more standard cell model, data points lie at the corners of cells and are shared between neighboring cells. At height one, a node covers (up to) a 2^k array of cells. The pointer refers to the minimum data point of this array. For 3D data, these 8 cells contain 27 data points. The data points along the exteriors of these cells are shared by neighboring cells. Assuming a trilinear function over the cell regions, the data field is continuous, so rendering at the deepest level of the tree can provide a more continous image than the previous methods.

Without compression, the spatial cost of this model, assuming floating point 3D data, is the size of the original data n plus approximately $n/7$ nodes each of which contains 8 trilinear coefficients, or $15n/7$. In many cases this extra spatial cost may not prohibitive.

Data compression is more complex for a cell model, because the data is shared between nodal regions. One solution is to separate the data into smaller grids which have redundant data points along the boundaries. For example, for a full 3D tree, each height 1 node could point to a cluster of 27 data points. In such a representation, the data level would contain approximately $27n/8$ data points, rather than n points, and the tree contains $n/7$ nodes. Larger clusters of 125 yield $125n/64$ data and $n/56$ tree nodes. But each tree node contains as many coefficients as needed by the model, e.g., 8 for the trilinear model. To achieve compression, a substantial fraction of clusters need to be discardable.

4.2.4 Continuity Issues

In choosing a model, one can give priority either to continuity between regions or to a best fit of the data over the defined region. We chose the latter, as being a more appropriate representation for scientific data. However, for some purposes, one might prefer a model that minimizes discontinuity along boundaries. For example, one could use a pyramid model where each pyramid level reduces the size of the data by two in each dimension, and stores either a mean or representative data point from the region below it. If the pyramids represented cells, imaging on a single level would give continuity. However, pyramids do not compress flexibly.

Another strategy would be to store a higher-order model, such as a tricubic, at each node. While this requires 64 coefficients rather than 8, it can also fit data more closely and provide continuity with neighbors. For some data sets, the extra compression that may be possible for a given amount of error may justify such a model. This can again provide continuity when imaging at a single level, as well as easy compressibility.

[2] Splatting ameliorates discontinutities, but introduces other rendering inaccuracies involving opacity [LH91].

However, even if continuity between regions on a particular hierarchy level is maintained, when imaging is done on different tree levels, discontinuities will result.

4.3 Evaluation Information

The two types of evaluation information stored in the hierarchy are nodal error and data importance.

4.3.1 Nodal Error

The nodal error is an average deviation of the model ($f(\mathbf{p})$) from the data ($g(\mathbf{p})$) within the region V (with $|V|$ data points) covered by the node. In the L_q norm, the equation is:

$$e = \left(\frac{1}{|V|} \sum_V |g(\mathbf{p}) - f(\mathbf{p})|^q \right)^{\frac{1}{q}}$$

For $q = 2$ (see Section 3.3), e can be computed from $\langle g, g \rangle$ and the coefficients of f; also, either a sum or an integral can be used. For $q \neq 2$, each node's e must be computed from scratch.[3] Experimental evidence, discussed later, suggests values of q much higher than 2 may give superior images for the same compression.

4.3.2 Data Importance

The second evaluation metric is data importance. In many data sets, different data values and/or different regions have different interest levels. For example, the air surrounding a CT scan may be considered unimportant, or certain values in a simulation may be known to represent background. Initially all data is given importance 1. Using an interactive transfer function editor, the user can design an importance function giving each data value an importance between 0 and 1. At each tree node the maximum importance of any data point in its region is stored (in one byte).

5 Selective Traversal

Once the tree has been created, the user can selectively traverse it either for imaging or for data compression. A number of evaluation parameters are used to control the traversal, most of which are calculated on the fly not stored in the tree. Left in their default state, the parameters have no effect and traversal continues until a node with no error is found. The user-controlled evaluation parameters are:

1. *Model Error Threshold*: The user sets an allowed error value between zero and one, which is multiplied by the standard deviation of the entire volume. Nodes with *nodal error* (possibly modified below) at or below this threshold are rendered as single objects.

2. *Data Importance*: Using data importance is optional. When activated, the node's *nodal error* is multiplied by its *importance* before being compared to the threshold.

3. *Pixel Coverage Weighting*: The user can define a pixel coverage value such that any node that projects to less than the coverage is given reduced importance. The *nodal error* is again multiplied by this value before being compared to the threshold.

4. *Region Restrict*: The user can interactively define a rectangular 3D region and limit traversal to nodes and data within the region.

5. *Dimension Restrict*: For data with greater than three dimensions, the user can define which three should be used for imaging, as well as the constant values for the dimensions not imaged. By default, imaging uses the first three dimensions and plane 0 of any further dimensions.

[3] except $q = \infty$, the max-norm

6. *Tree Depth*: The user may specify a depth in the tree such that traversal never goes deeper.

7. *Allowed Time*: This option is somewhat orthogonal to the above. If set, the system calculates the deepest level that, using a rendering cost estimate, can be rendered in the allowed time.

8. *Clipping*: When traversing for visualization, there is a further automatic evaluation parameter of visibility. Each node is checked to see if the region it represents is at all visible on the screen, and returns immediately if not.

The evaluation metrics defined above are checked at each node of the tree during selective traversal to determine whether traversal should descend further or stop and return. If traversal stops and imaging is requested, the region is drawn if visible. If the traversal is for compression, this node becomes a leaf; its child pointer is set zero, and the rest of the subtree is discarded.

5.1 Selective Traversal for Visualization or Compression

Selective traversal makes it possible to replace the data with a hierarchical representation that adequately represents it. In many cases, this representation is smaller than the original data set, because some data values or regions are of no importance, because the data values in some regions are constant or otherwise modeled very accurately, or because some amount of error compared to the original data is tolerated. In these cases, the tree representation can be written out and used in future to represent the data.

When selective traversal is used for imaging, the user will often allow much greater error, in the interest of speed, than she would as a permanent data representation. For such a use, the hierarchy must be retained because future more accurate images may require it.

6 Rendering Methods

Our hierarchical approach is not restricted to any particular rendering method. The implementation performs direct volume rendering using the coherent projection approach [WVG91]. This method calculates information concerning the projection of a rectilinear cell and uses hardware Gouraud-shading for rapid rendering. It is generally used with orthogonal projection on rectilinear cells. While the method does not produce the highest quality images, it does produce quite good images rapidly.

Consider renderings with no compression on a 3D volume of resolutions (r_x, r_y, r_z), with $n = r_x r_y r_z$. The voxel mean model treats the projected region as having a constant value. One region drawn for each data point, n in all. The voxel and cell trilinear models treat the projected region as a trilinear function, which is evaluated at region corners. Voxel trilinear draws about $n/8$ cells, most covering 8 voxels. Cell trilinear draws $(r_x - 1) * (r_y - 1) * (r_z - 1)$ cells. Thus, the voxel models and cell models do not align exactly. Constant value coherent projection is approximately twice as fast as when corner values vary, due to reduced amounts of interpolation.

Splatting [Wes90, LH91] provides a fast, reasonable alternative to rendering constant value voxels. We preferred coherent projection because the region projections fit more continuously than do splats, particularly when neighboring regions are of different sizes, due to compression. Higher-quality rendering methods, including ray-casting or software projection methods, could be used with the hierarchical approach as well.

It should be pointed out that the evaluation parameters, except for coverage, do not take into account imaging issues. For example, the error does not consider the transfer function used to map data values to colors; nor does it explicitly take into account possible discontinuity between neighbor regions that may be drawn in one image. The hierarchical model could accommodate a more image-based metric than we have used, should this be considered more important.

7 Error Analysis

Two related but not identical issues must be considered in examining the hierarchical approach, or any visualization method. The first is the validity of the representation compared to the actual data being used. The second is the quality of the image, a more difficult thing to measure.

Our basic metric for data validity is the nodal error. In selectively traversing the tree, this is weighted, if the user wishes, by parameters described previously. The user-specified allowed nodal error determines how closely the data representation used for visualization or compression fits the original data.

Our metric for image quality is the closeness of the resultant image with some weighted error to a *standard image* produced by the same visualization method allowing no error. Thus, it is better to refer to this comparison as the *difference* between images rather than the error. However, since the comparisons use techniques of error analysis, it seems more natural to retain the name error for the analysis. The visualization methods are constant-value coherent projection for the mean voxel model and varying-field coherent projection for the trilinear models.

In judging image quality we attempt to quantify what is partly a subjective evaluation. To do so, we have examined five *error levels* determined by the variations between the standard image and those produced with varying error and other evaluation parameters. The five error levels are quantified by their root mean squared image errors (RME2), and their absolute maximum image error (MAE) compared to the standard image. By image error, we mean pixel-by-pixel color difference between the standard image and the image with error, scaled to the range 0-255. Only pixels that are non-black in at least one image count toward the RME2. Difference images show the absolute value of the difference between two images.

The five levels, with subjective evaluations, are specified as follows:

1. *Level 0: RME2 = 0; MAE = 0.* This is the standard image with no error.
2. *Level 1: RME2 < 3; MAE < 10.* A level 1 image is nearly indistinguishable from the standard.
3. *Level 2: RME2 < 6; MAE < 20.* A level 2 image is still very close to the standard, though the difference image does show a slight but visible difference. Differences between images are subtle.
4. *Level 3: RME2 < 9; MAE < 40.* A level 3 image is close the original on first glance, but on closer examination differences can be seen. Differences are obvious on difference images.
5. *Level 4: RME2 < 12; MAE < 80.* A level 4 image shows clear differences and sometimes blocky artifacts, but the main features visible in the standard image are still clear.

Images with even greater error may be useful for quick positioning and scanning. We were only interested in evaluations that provided images with good information content. The above criteria gives considerable weight to the maximum absolute difference, although this may involve only a few pixels. We chose to do so because in scientific data this seemed an important consideration. For images mainly of aesthetic value, the weighting for MAE could probably be reduced with little ill effect.

8 Experimental Results

In our experiments, we concentrated on answering three main questions:

1. What are the space and time costs of using a hierarchy compared to rendering from the data itself?
2. How much can this be reduced using lossless and lossy compression?
3. What choice of evaluation parameters provides a good balance between imaging time and quality?

Dim.	Data Set	Resolution	Sample Points	Data Type	Std.Dev
3D	Hipiph	64x64x64	262,144	float	.01845
	Sod	97x97x116	1,091,444	byte	15.62
	P6985	244x91x64	1,421,056	float	.004208
	Dolphin	320x320x40	4,096,000	short	612.3
	CTHalf	251x512x113	14,521,856	short	612.6
	Mandelbrot Set	256x256x256	16,777,216	float	443.98
	CTHead	512x512x113	29,622,272	short	564.1
4D	Radm	29x69x63x9	1,134,567	float	.001014
	Heart	256x256x8x16	8,388,608	short	57.38

Table 1: Data Set Characteristics

We especially wished to explore ways to quantify our results. Space limitations force this description to be abbreviated; a longer version is available from the authors.

We used a selection of 3D and 4D data sets, including molecular simulations, CT data, simulated CFD data, mathematical functions, and experimentally measured pollution data.[4] All were on rectilinear grids. Table 1 shows characteristics of the data sets we explored. Statistics were run on a Silicon Graphics Reality Engine II, with 64 megabytes of memory.

8.1 Space Usage

This section examines the spatial requirements of using hierarchies, considering lossless and lossy representations. We assume, for ease of comparison, that all our data was floating point (4 bytes). All mean nodes require 16 bytes, and all trilinear nodes require 44 bytes.

8.1.1 Three-Dimensional Data

As Table 2 shows, for 3D data, the mean and voxel trilinear hierarchical representation without compression took about 1.57 times the size of the data and cell trilinear took 2.57 times that size. To measure compression from these sizes we used the size of the tree actually traversed plus the size of the data actually accessed, assuming the rest could be discarded. For the cell trilinear model, either the entire data set must be kept, or clusters of 27 data points involving considerable redundancy must be kept, whichever yields the lower total.

In general, the hierarchical representation with no error allowed little compression. The exceptions were the CTHead, CTHalf, and Mandelbrot Set data sets, which had substantial constant regions.

Allowing even 1% error (relative to the dataset's standard deviation) usually brought the required size down below the size of the original volume, and 5% was commensurably more successful.

On the data sets where importance and restriction were used, these generally brought the required hierarchy size down to a fraction of the original data size, even allowing no error on the data that remained. Allowing more error on those volumes had relatively little impact.

8.1.2 Four-Dimensional Data

For 4D data and hexadec trees, the voxel mean model took about 1.27 to 1.33 times the size of the data, compared an optimal ratio of 1.267. We also compared the size taken by the 4D hexadec tree to separate 3D trees for each time slice. In all cases Table2 considerable savings.

[4] Sources omitted as requested in call-for-papers

Data Set	Data Model	Std	No Error	1% Error	5% Error	+ Importance/Restrict No Error	1% Error	5% Error
3D Data								
Hipip	Voxel Mean	1.57	1.57	1.03	0.55	-	-	-
	Voxel Trilinear	1.57	1.57	0.59	0.27	-	-	-
	Cell Trilinear	2.57	2.57	1.21	0.42	-	-	-
Sod	Voxel Mean	1.59	1.12	1.11	1.01	0.34	0.34	0.34
	Voxel Trilinear	1.59	1.46	1.44	1.31	0.55	0.55	0.55
	Cell Trilinear	2.59	2.49	2.45	2.23	1.03	1.03	1.00
P6985	Voxel Mean	1.58	1.58	1.44	0.48	-	-	-
	Voxel Trilinear	1.58	1.58	0.46	0.14	-	-	-
	Cell Trilinear	2.58	2.56	0.76	0.19	-	-	-
Dolphin	Voxel Mean	1.57	1.26	0.36	0.17	0.21	0.17	0.08
	Voxel Trilinear	1.57	1.27	0.42	0.24	0.12	0.11	0.06
	Cell Trilinear	2.57	2.28	0.76	0.50	0.65	0.37	0.19
CTHalf	Voxel Mean	1.58	0.87	0.79	0.48	-	-	-
	Voxel Trilinear	1.58	0.95	0.92	0.58	-	-	-
	Cell Trilinear	2.58	2.55	1.78	0.93	-	-	-
Mandelbrot Set	Voxel Mean	1.57	0.87	0.79	0.48	-	-	-
	Voxel Trilinear	1.57	0.94	0.77	0.62	-	-	-
	Cell Trilinear	2.57	1.96	1.77	1.06	-	-	-
CTHead	Voxel Mean	1.58	0.73	0.68	0.43	-	-	-
4D Data								
Radm	Voxel Mean	1.33	1.14	0.71	0.44	-	-	-
	Separate 3D Trees	1.62	1.41	-	-	-	-	-
Heart	Voxel Mean	1.27	1.26	1.26	1.26	0.85	0.85	0.85
	Separate 3D Trees	1.57	1.57	-	-	-	-	-

Table 2: Space Usage by Test Data Sets, discussed in Section 8.1. Numbers represent the ratio of the necessary space for the data representation divided by the size of the original data set. Assumes data values take 4 bytes, mean nodes take 16 bytes, and trilinear nodes take 44 bytes. "+ Importance/Restrict" means restriction and/or importance was used.

8.2 Image Evaluation and Selective Traversal

For this exploration, we compared images generated with varying amounts of error to those made with no error for their particular data model (see Table 3). The times are given in c.p.u. seconds using one processor.

In all cases, we found noticeable speedup (usually by two or three times) between the image generated at "level 0" (no error), compared to "level 1" error, from which it is is virtually indistinguishable. (Error "levels" were defined in Section 7.) In going from level 0 to level 2 error, the speed-up was from three to ten times, and images were generally nearly indistinguishable again. Level 3 images, where small differences begin to appear, saw speed-ups, compared to level 0, of from four to twenty times. Even level 4 images are hard to distinguish from the standard on many volumes, but can be drawn, usually, 10 to 50 times faster. Quite reasonable images at greater error levels can often be drawn hundreds of times faster.

Data Set	Model		No Tree	Level 0 RME2=0 MAE=0	Level 1 RME2≤3 MAE≤10	Level 2 RME2≤6 MAE≤20	Level 3 RME2≤9 MAE≤40	Level 4 RME2≤12 MAE≤80
3D Data								
Hipip	V.Mean	Time	11.71	11.98	5.99	1.52	0.52	0.24
		UserE	0.00	0.00	0.018	0.15	0.25	0.90
	V.Tril.	Time	22.01	3.95	1.33	0.84	0.38	0.25
		UserE	0.00	0.00	0.02	0.05	0.20	0.25
	C.Tril.	Time	22.08	27.96	4.91	2.66	1.02	0.44
		UserE	0.00	0.00	0.015	0.035	0.12	0.25
P6985	V.Mean	Time	63.29	68.22	41.78	21.31	15.61	7.08
		UserE	0.00	0.00	0.022	0.040	0.055	0.11
	V.Tril.	Time	128.57	22.17	11.4	9.04	5.78	3.48
		UserE	0.00	0.00	0.003	0.0055	0.014	0.032
	C.Tril.	Time	128.57	151.45	65.21	49.26	17.74	8.29
		UserE	0.00	0.00	0.0012	0.0023	0.013	0.03
CTHalf	V.Mean	Time	391.56	332.23	168.10	150.77	145.69	80.78
		UserE	0.00	0.00	0.05	0.075	0.10	0.20
	C.Tril.	Time	961.82	778.02	548.37	476.82	307.75	113.07
		UserE	0.00	0.00	0.01	0.018	0.04	0.12
CTHead	V.Mean	Time	799.1	567.9	357.4	196.2	129.6	77.0
		UserE	0.00	0.00	0.1	0.2	0.30	0.45

Table 3: Time and Error by Test Data Sets, discussed in Section 8.2. Times are c.p.u. seconds on Reality Engine II.

Figures 6, 7, 8 and 9 (see slides) shows images of error level 0, 2, and 4 (at the settings used for Table 3) for several volumes using the voxel mean and/or cell trilinear models. They also show a difference image between levels 0 and 4, with color differences multiplied by 5 for greater visibility.

We found the described levels of image difference to be an interesting first step in quantifying image quality, but don't feel it is really comprehensive. Other characteristics of the image, which we are not using to control rendering, clearly play a major role in image quality. For example, the small Hipip volume shows the clearest differences in image quality using the five levels, and also the greatest speed up. But for the P6985 flow volume (photos, Figure 5) and the CTHalf (slides, Figure 8 and Figure 9), level 4 images were visually difficult to differentiate from level 0 images, even on the monitor. For P6985, this is probably related to the relatively smooth gradient of the data. For the relatively large CTHalf volume drawn within the window, regions projected to few pixels, so discontinuities and differences are hard to see.

Figure 11 compares voxel mean images to cell trilinear images with no error (left) and requiring the same rendering time (right), showing that cell visualization can be noticeably more continuous.

8.2.1 Error Exponents

For our final examination, we explored raising the nodal error to higher exponents (Section 4.3.1). This gives greater relative weight to large errors. Figure 10 (see slides) compares a zoomed image of the Hipip volume using no error (time 8.36 seconds) to one using the standard L_2 error, the L_6 error, and L_{12} error. All these took 0.24 seconds to draw and accessed almost the same amount of the tree. But due to different error criteria, the various methods made different choices on which *parts* of the tree to access. All would be classified, by our system, as level 4 errors. Notice that with the L_2 error, some of the small red and blue features have disappeared, but gradually reappear with the higher error exponents. Some, however, did not reappear at this level of error for any error exponent.

The cell trilinear method did not show much sensitivity to the choice error metric on this data set.

8.2.2 General Observations

In studying images from the different data models, we found the voxel mean model generally more successful than we expected. Sometimes (not always), in nearly front-on views, discontinuities are obvious. Allowing greater amounts of error, discontinuities between regions were often less using the mean model than the trilinear ones.

The voxel trilinear model could sometimes produces images very close to the voxel mean model is much less time, but it was inconsistent. Because the trilinear function extrapolates the function over the data points, it can cause irritating discontinuities between neighboring regions when the extrapolations do not match. It would be possible, though we did not implement it, to image using constant value voxels from the voxel trilinear model. This might make it possible to keep the advantages of both.

The cell trilinear model, because the image drawn at the lowest level was continuous, did produce the most consistently pleasing images. That is, on volumes where the best possible voxel image tended to appear discontinuous, this model did not. There was, though, a commensurate cost in extra storage that might not be worth it on large volumes where differences between the images are often slight anyway. Interesting, discontinuities between regions were occasionally obvious when error was allowed, showing the cost of attempting to best fit the data throughout the region rather than only along the borders. For scientific applications, we still feel a good data model is better than a good image.

9 Conclusions

We found that the hierarchical strategy was extremely successful in providing a flexible imaging approach. It provides a number of easy-to-use parameters that control the image quality and speed in an intuitive manner. Because the parameters are related to the error compared to the original volume, they allow users to control the accuracy of the image. While hierarchies do not compress as well as other methods, the compressed version can be used nearly as quickly as the original data, the only decompression cost being involved in tree traversal and evaluation. Further, in many cases, users may feel more confident than we did in using restriction and importance to reduce the necessary data size.

We believe the hierarchical approach could be extremely helpful for irregularly sampled data sets. The use of an error-controlled regular hierarchy avoids many of the problems in imaging irregular regions, and if many data points are clustered in small regions, this regions can be given less weight when they project to only a few pixels. We also think it would be interesting to extend this work in areas other than scientific visualization, using metrics more closely tied to image quality.

References

[Cha93] Judy Challinger. Scalable parallel volume raycasting for nonrectilinear computational grids. In *IEEE Parallel Visualization Workshop*, October 1993. (to appear).

[Chu92] C. K. Chui. *An Introduction to Wavelets.* Academic Press, Inc., 1992.

[DFM87] Robert A. Drebin, Elliot K. Fishman, and Donna Magid. Volumetric three-dimensional image rendering: Thresholding vs. non-thresholding techniques. *Radiology*, 165:131, 1987.

[FS93] Thomas Funkhouser and Carlo Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (ACM Siggraph Proceedings)*, 27(4):247–254, August 1993.

[GSCH93] Steven J. Gortler, Peter Schroeder, Michael F. Cohen, and Pat Hanrahan. Wavelet radiosity. *Computer Graphics (ACM Siggraph Proceedings)*, 27(4):221–230, August 1993.

[Lev89] Marc Levoy. Design for a real-time high-quality volume rendering workstation. In *Volume Visualization Workshop*, pages 85–90, Chapel Hill, NC, May 1989. Dept. of Computer Science, University of North Carolina.

[Lev90] Marc Levoy. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications*, 10(2):33–40, March 1990.

[Lev92] Marc Levoy. Volume rendering using the fourier projection-slice theorem. In *Proceedings of Graphics Interface '92*, Vancouver, B.C., 1992. Also Stanford University Technical Report CSL-TR-92-521.

[LH91] David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 25(4):285–288, July 1991.

[Mal89] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[Mal93] Tom Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.

[Mea82] Donald J. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19:129–147, 1982.

[Mur93] Shigeru Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics and Applications*, 13(4):50–56, July 1993.

[NH92] Paul Ning and Lambertus Hesselink. Vector quantization for volume rendering. In *1992 Workshop on Volume Visualization*, pages 69–74, Boston, Mass., October 1992. ACM.

[NH93] Paul Ning and Lambertus Hesselink. Vector quantization for volume rendering. In *Visualization '93*, San Jose, Ca, October 1993. IEEE.

[ST90] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, December 1990.

[VGW93] Allen Van Gelder and Jane Wilhelms. Rapid exploration of curvilinear grids using direct volume rendering. In *Visualization 93 Conference*, San Jose, CA, October 1993. IEEE. to appear.

[Wes89] Lee Westover. Interactive volume rendering. In *Volume Visualization Workshop*, pages 9–16, Chapel Hill, NC, May 1989. Dept. of Computer Science, University of North Carolina.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–76, August 1990.

[Wil92] Peter Williams. Interactive splatting of nonrectilinear volumes. In *Visualization '92*, pages 37–44. IEEE, October 1992.

[WVG91] Jane Wilhelms and Allen Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics (Proceedings ACM Siggraph)*, 25(4):275–284, 1991.

[WVG92] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992. Extended abstract in ACM Computer Graphics 24(5) 57–62; also UCSC technical report UCSC-CRL-90-28.

Figure 4: The Radm 4D data set showing the YZT dimensions for X=5. T varies upward in the image.

Figure 5: The P6985 data set showing voxel mean, voxel trilinear, and cell trilinear with no error (level 0) descending in the left column, and the same models with level 4 error in the right column. The fastest image shown was 44 times faster than the slowest.

The following are shown as slides:

Figure 6: The Hipip data set using the voxel mean model showing error levels 0, 2, and 4 and the differences of levels 0 and 4 scaled by 5.

Figure 7: The Hipip data set using the cell trilinear model showing error levels 0, 2, and 4 and the differences of levels 0 and 4 scaled by 5.

Figure 8: CT Half data set using the voxel mean model showing error levels 0, 2, and 4 and the differences of levels 0 and 4 scaled by 5.

Figure 9: CT Half data set using the cell trilinear model showing error levels 0, 2, and 4 and the differences of levels 0 and 4 scaled by 5.

Figure 10: The Hipip data set using the voxel mean model showing a no error images compared to level four images with error exponents of 2, 6, and 12. Notice the features that disappear at error exponent 2 and some of which reappear at higher error levels.

Figure 11: The Radm data set comparing the mean and cell trilinear models. Left top is the no error voxel mean image; left bottom, the no error cell trilinear image; and at the right, images using those respective models with some error, taking 1.68 seconds to draw.