# An Octree-based Multiresolution Approach Supporting Interactive Rendering of Very Large Volume Data Sets

D. Pinskiy
CIPIC*
Department of Computer Science
University of California
Davis, CA, USA

E. Brugger
Lawrence Livermore National Lab.
Livermore, CA, USA

H. Childs
Lawrence Livermore National Lab.
Livermore, CA, USA

B. Hamann
CIPIC*
Department of Computer Science
University of California
Davis, CA, USA

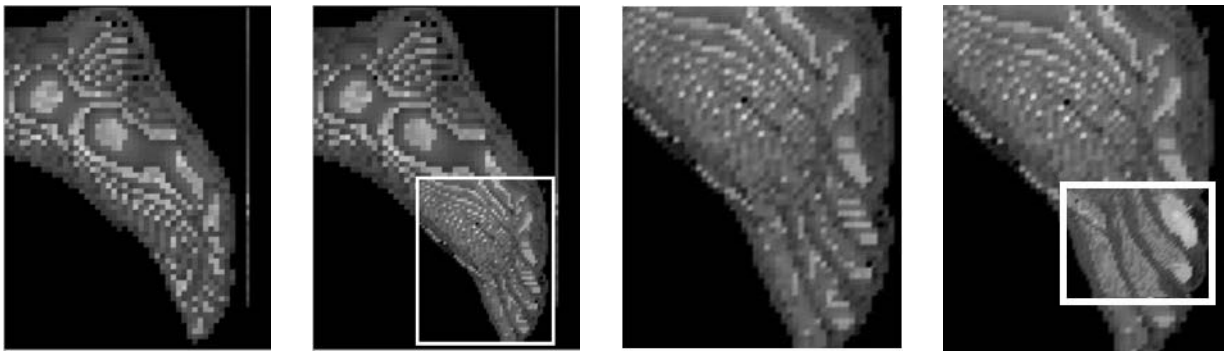

Figure 1: Local refinement of nested ROIs in foot data set.

**Abstract** *We present an octree-based approach supporting multiresolution volume rendering of large data sets. Given a set of scattered points without connectivity information, we impose an octree data structure of low resolution in the pre-processing step. The construction of this initial octree structure is controlled by the original data resolution and cell-specific error values. Using the octree nodes, rather than the data points, as elementary units for ray casting, we first generate a crude rendering of a given data set. Keeping the pre-processing step independent from the rendering step, we allow a user to interactively explore a large data set by specifying a region of interest (ROI), where a higher level of rendering accuracy is desired. To refine an ROI, we are making use of the octree constructed in the pre-processing step. Our approach is aimed at minimizing the number of computations and can be applied to large-scale data exploration tasks.*

*Keywords:* Multiresolution, local refinement, volume rendering, octree

## 1 Introduction

Ray Casting is a fundamental visualization method for volumetric data sets. It produces realistic looking images, but it has a significant drawback – its time complexity. In most scientific fields, data sets have grown in size to several million points. Considering finite el-

*Center of Image Processing and Integrated Computing

ement/difference simulations, ray casting can become extremely expensive. Moreover, producing high-detailed renderings can be wasteful if a user is interested in only a small portion of a large data set.

We deal with the "massive data problem" by applying different levels of detail for data representation. Using a hierarchy constructed in a pre-processing step, we generate a coarse approximation that serves preview purposes for large-scale data sets of scattered points. We refine this approximation until it becomes sufficiently detailed for a user to identify an ROI. Once an ROI has been identified, we locally refine this region and generate a much more accurate data representation inside the ROI. If even more precise accuracy is needed inside the ROI, nested ROIs are used, see Figure 1.

## 2 Related Work

In the conventional ray casting approach based on alpha blending, the fundamental formula that is used for accumulating "illuminated material samples" along a ray from (sample) point 1 to point N is based on linear blending, defined by the formula

$$B_i = A_{i+1}C_{i+1} + (1 - A_{i+1})B_{i+1}, \qquad (1)$$

where $A_{i+1}$ is the alpha value of the (i+1)-th point, $C_{i+1}$ is the function value (color) of the (i+1)-th point, $B_i$ is the resulting color seen at the i-th point, and $B_{i+1}$ is the background color of the (i+1)-th point – the resulting color seen at the (i+1)-th point [1].

Many techniques have been proposed to improve the efficiency of conventional ray casting, which has complexity $O(N^3)$. Levoy suggested to terminate ray traversal when accumulated color reaches some preset value [2]. This ray termination condition, which we incorporated in our method, is given by

$$(1 - A_1)(1 - A_2)...(1 - A_i) < e, \qquad (2)$$

where $e$ is some threshold value.

Another improvement of the ray-casting algorithm is based on skipping "empty regions" in space when traversing along a ray. Wan et al.[3] suggested to find approximated boundaries of "objects" in a pre-processing step and then traverse rays only inside the objects. Unfortunately, their approach introduces substantial overhead, especially if a user is only interested in a small part of an object, or an object is large relative to the bounding box of the entire data set. An efficient technique was developed by Levoy [2]. He employed a pyramidal data representation to encode spatial coherence. However, his algorithm is designed for rectilinear data, not for arbitrarily scattered points. Levoy's method does not directly support local refinement, and it can lead to high memory requirements.

We now summarize our method. Considering some error condition, we first construct an octree representation similarly to the approach described by Laur et al [4]. Using an octree-based approximation, one generates a data-dependent, spatial decomposition that adapts to the underlying complexity of the data. The technique described by Hamann [5], for example, is aimed at eliminating points in nearly linearly varying regions.

In principle, our technique is related to the idea of constructing a "data pyramid" – a data hierarchy of rectilinear (hexahedral) cells with increasing precision [6]. So-called multiresolution methods have been developed for polygonal and polyhedral approximations of surfaces and graphs of bivariate functions defined over volumetric domains. Such approaches were described by Eck et al. [7], for example. Our data-dependent, octree-based method can be viewed as a semi-automatic and user-driven hierarchical scheme, supporting the visualization of large-scale scientific data by multiple levels of approximation.

## 3 Octree Construction

We assume that the input to our method is a set of scattered data: points in space with associated (scalar) function values without known connectivity information. In a pre-processing

step, we construct an octree in the following way: First, we compute the bounding box of all original points. This bounding box is then subdivided in an octree fashion until each leaf node in the octree satisfies a specific termination criterion for refinement. We consider two types of termination criteria: (i) the number of original points lying in it is smaller than some threshold number or (ii) the error between the function values of the original points inside the node and an average function value associated with the node is smaller than some error threshold. The second termination criterion helps us to identify regions where scattered points have nearly the same function values. In such regions, we can reduce storage requirements substantially by representing the given scalar field at a coarse resolution.

Our octree pre-processing step is independent of the orientation of the eventually chosen viewing plane. Therefore, once the octree is constructed, a user can interactively adjust the location of the viewing plane without degrading the performance.

# 4 Generating Approximations

We can utilize the octree data structure to generate a first preview of a data set. We process the octree's nodes that lie no deeper than in the K-th level of the tree. As a result of our octree construction, all points inside the same node have nearly the same function values, and, introducing little error, we can approximate these points by a single point (node center) with average function and alpha values of all points inside the node. Since we are generating a crude approximation, we can assume that the K-th level is relatively "low" in the octree. Thus, most nodes in the K-th level should contain large numbers of original points. As a result, to traverse a single ray, we only have to use a relatively small number of large integration spans. These spans are defined by ray-node intersection points.

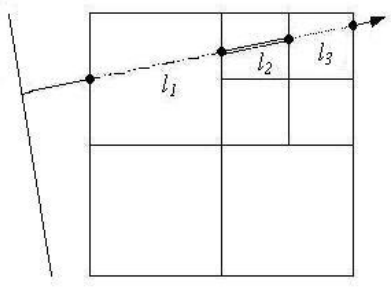


Figure 2: Ray traversing through quadtree cells.

## 4.1 Adjusting Ray Casting

Since we use variable-size octree nodes, we cannot apply the conventional ray casting formula directly. This formula assumes that the influence of each voxel on the corresponding resulting pixel in the image plane depends only on the alpha value of the point and the point's position.

We need to take into account the lengths of the intersections of rays with the bounding boxes of octree nodes. A greater length of intersection results in a greater influence on the resulting pixel. Thus, we change the blending formula in the following way:

$$B_i = F(l_{i+1}, A_{i+1})C_{i+1} + (1 - F(l_{i+1}, A_{i+1}))B_{i+1}, \quad (3)$$

where $A_{i+1}$ and $C_{i+1}$ are average alpha and color values of points inside the (i+1)-th node; $F$ is a "weight function" that considers the length $l_{i+1}$ of a ray-box intersection and returns a weight corresponding to the intersection length; and $B_i$ is the resulting color seen at the i-th node. Figure 2 illustrates this formula for the 2D case. The resulting color $B_0$ is given as

$$B_0 = F(l_1, A_1)C_1 + (1 - F(l_1, A_1)) [F(l_2, A_2)C_2 + (1 - F(l_2, A_2))F(l_3, A_3)C_3].$$

## 4.2 Weight Function

Let us take a closer look at the weight function $F$'s arguments. The second function ar-

gument, opacity $A$, means this: Zero indicates an absolutely transparent object, and one corresponds to a completely opaque material. The first argument, length $l$, is more complicated. This length of intersection of a ray with an octree cell's bounding box is expressed not in units of length but in the number of data points that lie along the ray inside the cell. Calculating this number exactly is very time-consuming. However, we can approximate it if we assume that, on average, the data points are distributed uniformly inside the volume, and the distance between adjacent casting rays is equal to the average distance between adjacent data points. Suppose we have $n^3$ points inside the cell, aligned with the coordinate axes, then we assume $n$ points lie along a ray that intersects the cell and is parallel to one of the coordinate axes. For an arbitrarily oriented ray, the length of intersection can be calculated as

$$l = nd/edge, \tag{4}$$

where $d$ is the distance between the ray's entry and exit points in the cell, and *edge* is the length of the edge of the cell's bounding box.

We now describe the function by considering a simple case – a ray meets just one data point along its way inside the octree cell's bounding box ($l$ is equal to one). In this case, our adjusted formula (3) should yield the same result as the conventional ray-casting formula (1):

$$\begin{aligned} B_i = \quad & F(1, A_i)C_i + (1 - F(1, A_i))B_{i+1} = \\ & A_iC_i + (1 - A_i)B_{i+1}. \end{aligned} \tag{5}$$

A much more typical case is given when the number of points ($l$) that a ray "meets" is greater than one. Although all these $l$ points might have different function values, their values should be close since these points are enclosed in the same cell of an error-controlled octree. Therefore, their true function values can be substituted by the average value of the points inside the cell. Traversing the ray inside the cell would then be equivalent to applying the conventional ray casting to $l$ consecutive points that have the same function value, see

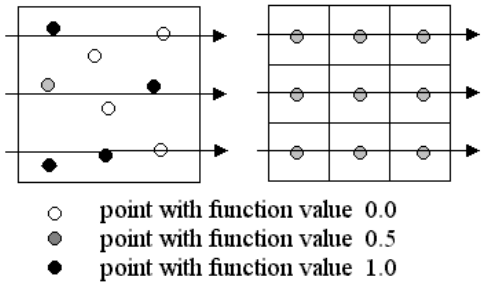


Figure 3: Approximation of ray-casting a set of scattered points (left) by ray-casting though consecutive rectilinear cells (right).

Figure 3:

$$\begin{aligned} & F(l, A)C + (1 - F(l, A))B = \\ & A_1C_1 + (1 - A_1)[A_2C_2 + (1 - A_2) \\ & [\cdots [A_lC_l + (1 - A_l)B]\cdots]], \end{aligned} \tag{6}$$

where $A_1 = A_2 = \cdots = A_K = A$ and $C_1 = C_2 = \cdots = C_K = C$. A specific function for which both (5) and (6) hold is

$$F(l, A) = 1 - (1 - A)^l, \tag{7}$$

see Appendix.

When $l$ is large, the brute force computation of $F(l, A)$ becomes rather expensive since the formula involves calculating the power of $l$. Fortunately, there is a way to approximate this formula. The graph of $F(l, A)$ has a distinct shape as shown in Figure 4. This function increases quickly in a range $[0; l']$. We can pre-compute $l'$ for different opacities and later, when $l > l'$, we can set $F(l, A)$ to one.

### 4.3 Traversal Procedure

For each ray, we call a recursive traversal procedure. The pseudocode for this procedure is given below. (To produce a coarse approximation, we process nodes that are no deeper than the K-th level; initially, the procedure's second argument N is set to the octree root, and the flag ENOUGH is set to False.)

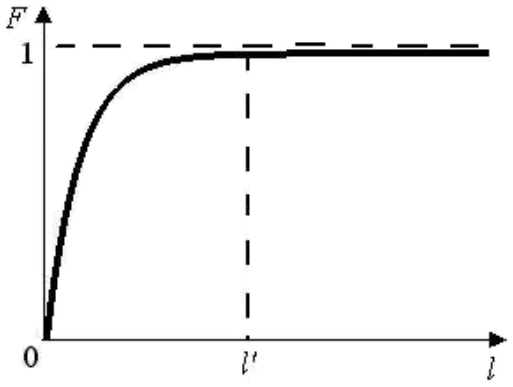


Figure 4: Graph of function $F(l, A)$ for particular $A$ value

```
Procedure Traverse(ray R, node N)
    If (R does not intersect N) Return;
    If (N is on K-th level of octree OR
        N is a leaf)
            Calculate color and opacity;
            If (Opacity threshold is reached)
                ENOUGH:=True;
    Else
        For each Child of N, starting from
            the child closest to the viewing
            plane, continuing with the next
            closest, etc. and ending with
            the farthest one
                Traverse(R, Child);
                If (ENOUGH) Return;
    Return;
```

### 4.4 Local Refinement

Once a sufficiently detailed initial image approximation is generated, a user can specify an ROI in it by using, for example, a bounding box. The coordinates of the ROI are used to identify the octree leaf nodes covering the ROI. These leaf nodes are then subdivided to achieve better image quality.

Our adaptive ray casting method supports both local refinement and coarsening operations [5]. Coarsening, in contrast to refinement, allows a user to merge leaf nodes into a larger one, thus reducing storage requirements. Coarsening is a means for accelerating the rendering process and keeping memory requirements for the octree low.

## 5 Implementation Details and Results

### 5.1 Distance Computation for Entry and Exit Points

An impelmentational issue that arises when using formulas (4) and (7) is how to efficiently calculate the distance ($d$) between the points where a ray enters and exits a node's bounding box. Using directly the Euclidean distance formula for two arbitrary points $A$=($x_0$, $y_0$, $z_0$) and $B$=($x_1$, $y_1$, $z_1$) in space, given by:

$$d = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}, \tag{8}$$

is computationally expensive. However, we can take advantage of the fact that we are dealing only with parallel projections and consider two cases: (i) A ray passes through opposite faces of a node's bounding box, and (ii) a ray passes through two faces sharing a common edge.

The first case can be easily detected by noticing that the ray's entry point has at least two coordinates that are equal to coordinates of the front-left-lower corner of the bounding box; and the exit point has, at least, two coordinates that are equal to coordinates of the rear-right-upper corner. Since we only consider parallel projections, all rays are parallel to each other. Therefore, all rays intersecting opposite faces of the same bounding box have the same length of intersection with this box. Let $d_i$ be the length of intersection of the ray with the bounding box of a node in the i-th level of the octree. This length $d_i$ is inversely proportional to the node's level in the octree, and it is given by

$$d_i = d_0/(i + 1), \tag{9}$$

where $d_0$ is the length of intersection of a ray with the bounding box of the entire data set. Thus, to compute $d_i$, we need to perform only two arithmetic operations.

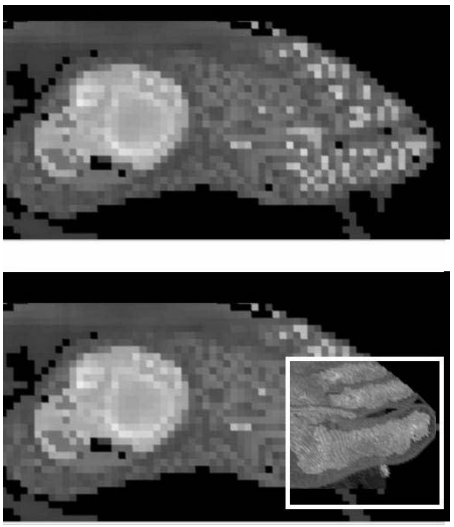

Figure 5: Local refinement of foot data set.

The second case is slightly more complicated. Since all rays are parallel, they form the same angles with the three coordinate axes. Let $\cos\alpha$ be the cosine of the angle between a ray and the X-axis; $\cos\beta$ be the cosine of the angle between a ray and the Y-axis; and $\cos\gamma$ be the cosine of the angle between a ray and the Z-axis. Depending on the entry point $a$ and the exit point $b$, the distance $d$ is defined by one of these three formulas:

$$d = (b.x - a.x)\cos\alpha, \tag{10}$$

$$d = (b.y - a.y)\cos\beta, \tag{11}$$

$$d = (b.z - a.z)\cos\gamma, \tag{12}$$

where $a.x$, $a.y$, and $a.z$ are the coordinates of the entry point $a$, and $b.x$, $b.y$, and $b.z$ are the coordinates of the exit point $b$.

### 5.2 Global and Local Octrees

We have implemented our volumetric data set representation as one global and one or more local octrees. The global octree represents the whole data set, and each local octree represents an ROI. Generally, the local octrees are not as deep as the global octree. However, since the local octrees cover less space, each of their nodes approximates less points than the global one does. As a result, a local octree yields a better approximation. Considering Figure 5, the main region is generated using the global octree with a depth seven, and the ROI is calculated using a local octree of a depth three.

Since constructing the global octree over the whole data set is a time-consuming task, we might pre-process the data set and construct a global octree prior to visualization. In contrast, building a local octree must be done in real time, since specifying and navigating an ROI is a part of the user-driven, interactive data exploration process. We can accomplish real-time local octree generation by using the main tree as a search tree to locate a node whose bounding box contains the ROI and then building the local octree considering only points that are inside the bounding box.

## 6 Conclusion

Employing an error-controlled octree data structure and a multiresolution approach, our algorithm allows a user to explore large-scale volumetric data sets by first creating a low-resolution approximation, and then locally refining regions of particular importance. This algorithm has great potential, as it supports high-speed rendering of multiple and even nested ROIs. We believe that adaptive approaches, like the one we have presented here, will play an increasingly important role in future large-scale scientific data exploration.

## 7 Acknowledgements

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; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

## References

[1] W. Schroeder, K. Martin, and B. Lorensen. *Visualization Toolkit.* 2nd edition, Prentice Hall, 1998.

[2] M. Levoy. Efficient Ray Tracing of Volume Data. *Transaction on Graphics,* 9(3):245–261, July 1990.

[3] M. Wan, A. E. Kaufman, and S. Bryson. High Performance Presence-Accelerated Ray Casting. *Proc. IEEE Visualization 1999*:379–386, 1999.

[4] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Proc. SIGGRAPH (Computer Graphics 25)*:285–288, 1991.

[5] B. Hamann. A Data Reduction Scheme for Triangulated Surfaces. *Computer Aided Geometric Design,* 11(2):197–214, March–April 1994.

[6] L. De Floriani. A Pyramidal Data Structure for Triangle-Based Surface Description. *IEEE Computer Graphics and Applications,* 9(2):67–78, March 1989.

[7] M. Eck, A. D. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. *Proc. SIGGRAPH (Computer Graphics 29)*: 173–182, 1995.

## Appendix

We prove by induction over the length of intersection $l$ this statement: Using the weight function

$$F(l, A) = 1 - (1 - A)^l \tag{13}$$

in the formula

$$B_i = F(l_{i+1}, A_{i+1})C_{i+1} + (1 - F(l_{i+1}, A_{i+1}))B_{i+1} \tag{14}$$

is equivalent to applying the conventional ray casting formula (1) to $l$ consecutive rectilinear cells that have the same function and opacity values.

For the base case ($l = 1$), we have

$$F(1, A) = 1 - (1 - A)^1 = A \Rightarrow$$
$$F(1, A)C + (1 - F(1, A))B =$$
$$AC + (1 - A)B.$$

We assume that the statement holds when the length of intersection is equal to 1, 2, $\cdots$, or $K$ data points:

$$F(K, A)C + (1 - F(K, A))B =$$
$$A_1C_1 + (1 - A_1)[A_2C_2 + (1 - A_2)$$
$$[\cdots[A_KC_K + (1 - A_K)B]\cdots]], \tag{15}$$

where $A_1 = A_2 = \cdots = A_K = A$ and $C_1 = C_2 = \cdots = C_K = C$.

Before we continue with the induction step, we note that the following equation holds:

$$A + F(K, A) - F(K, A)A = F(K + 1, A). \tag{16}$$

Now we perform the induction step ($l = K + 1$):

$$A_1C_1 + (1 - A_1)[A_2C_2 + (1 - A_2)$$
$$[\cdots[A_{K+1}C_{K+1} + (1 - A_{K+1})B]\cdots]] =$$
$$AC + (1 - A)[F(K, A) + (1 - F(K, A))B] =$$
$$AC + F(K, A)C + (1 - F(K, A))B -$$
$$F(K, A)AC - (1 - F(K, A))BA =$$
$$(A + F(K, A) - F(K, A)A)C +$$
$$(1 - (F(K, A) + A - F(K, A)A))B =$$
$$F(K + 1, A)C + (1 - F(K + 1, A))B$$

That concludes our proof.