Adaptive Error Bracketing for Controlled-Precision Volume Rendering[1]

Kevin Novins, James Arvo, David Salesin
Program of Computer Graphics
Cornell University
Ithaca, New York 14853
November 1992

Abstract

We present a new ray tracing approach to volume rendering in which the low-albedo volume rendering integral for each ray is efficiently computed to any prescribed accuracy. By bracketing the emission and absorption functions along each ray with adaptively refined step functions, computation is directed toward large sources of error and continued until a desired accuracy is reached. As a result, coarse approximations can be used in regions that are nearly uniform, of low emission, or of low visibility due to absorption by material closer to the eye. Adaptive refinement for each ray is performed using a hierarchical organization of the volume data; at each step, a part of the ray estimated to contribute large error is refined, and the approximate integral is updated incrementally. Our current implementation operates on regularly-spaced data samples combined with trilinear interpolation; however, the concepts described apply to more general data topologies and reconstruction filters.

1 Introduction

Approximation is a powerful tool for volume rendering. Carefully chosen approximations can lead to dramatic speed-ups by allowing closed-form evaluation of integrals and by taking advantage of special-purpose graphics hardware. Unfortunately, few of these approximations readily admit error metrics, which makes controlling image fidelity difficult at best. A good error metric also has an extra benefit: by providing reliable bounds on accuracy, it may allow an algorithm to concentrate its effort where the error reduction is

[1]Technical Report TR92-1312, Cornell University Department of Computer Science, Ithaca, New York 14853 USA.

largest, thereby affording an additional speed-up. In this paper, we describe an algorithm that makes use of reliable bounds to give substantial speed-ups for a certain class of volume data and level of approximation.

Approximations in volume rendering fall into two categories [15]: approximations in *reconstruction* and approximations in *projection*.

Reconstruction describes the process of extending point-sampled data to a continuous three-dimensional scalar field. As an approximation to the ideal but impractical sinc filter, reconstruction is commonly performed using trilinear or Gaussian interpolation. Other approximations include "nearest neighbor" [12] and combinations of bilinear and linear interpolation [9].

The role of projection is to create a two-dimensional image by integrating the reconstructed field in depth. Projection is accomplished in a variety of ways. One approach is to ray trace the image, integrating along each ray from front to back until the accumulated opacity exceeds a given threshold [6, 7]. This approach has two sources of error: first, the numerical integration contributes error that is neglected; and second, if no provision is made to measure the effect of the remainder of the ray, then this unintegrated part can in theory contribute error whose magnitude is arbitrarily large, no matter how high the opacity of the integrated part of the ray.

Other projection methods have been devised to take advantage of special-purpose graphics hardware, thus substantially increasing rendering speed. For example, in "splatting" [15, 16], individual samples are reconstructed and projected in screen space via compositing operations. These operations introduce a number of errors that are difficult to characterize—for example, the errors due to neglecting overlap in depth of the filter kernels. Another algorithm that takes advantage of graphics hardware is the "coherent projection" approach of Wilhelms and Van Gelder [17]. Although empirical estimates of error are given by the authors, it is unclear whether an analytic metric for this projection technique also exists.

As an algorithm for approximate volume rendering, the "hierarchical splatting" technique of Laur and Hanrahan [5] is interesting in that it provides a tunable accuracy control by allowing the coarseness of the data approximation to vary. This control effectively allows for a kind of time–accuracy tradeoff. However, the algorithm does not provide any control over quality in an absolute sense. Indeed, even the "highest quality" renderings produced by the algorithm contain significant errors due to the inherent inaccuracies of splatting.

Novins and Arvo [11] explore an approximation to the volume rendering integral that provides absolute bounds on the error due to projection of a single voxel along a ray. However, the technique, while allowing for renderings of arbitrarily high accuracy, does not provide for a very good time–accuracy tradeoff, in that the difference in computation time for the highest and lowest precision results is small.

In this paper, we describe a new approach to the projection problem that provides absolute error bounds, and that allows the competing goals of speed and accuracy to be traded off in a reasonable way. The algorithm works by subdividing the integral along each ray into small regions, bracketing the contribution of each region, and refining the regions adaptively until a prescribed error tolerance is reached.

The algorithm as described in this paper makes use of an accurate integral evaluation routine, such as the one described by Novins and Arvo [11], to speed convergence. However, any such technique giving accurate error bounds could be used. Moreover, the algorithm can be made to work without using any explicit evaluator; however, convergence is slower.

Finally, the algorithm as described here does not address error due to reconstruction. Although a trilinear reconstruction technique is assumed here, the algorithm can accommodate any continuous reconstruction filter.

In the next section, we express the problem of approximating the volume rendering integral in terms of interval arithmetic. From this formulation we derive an algorithm for bracketing the continuous integrand by upper and lower step functions, and for adaptively refining these functions wherever the error is estimated to be large.

2 Formulation

Our algorithm is based on a simple model of volume illumination that was originally developed as a model of radiative transfer [3, 13]. The model is physically-based and allows for independent absorption and emission functions throughout a volume.

The volume illumination model can be expressed as a set of one-dimensional integrals, with each integral representing the intensity of light emanating from the volume along a single ray. For each ray originating at the eye, the total intensity $I(a,b)$ emanating toward the eye from an absorbing and emitting medium between points a and b is given by

$$I(a,b) \;=\; \int_a^b T(a,t)\,\epsilon(t)\,dt, \tag{1}$$

where $T(a,t)$ is the *transparency* of the medium between a and t, given by

$$T(a,t) \;=\; e^{-\int_a^t \alpha(s)\,ds}, \tag{2}$$

and $\alpha(x)$ and $\epsilon(x)$ are the respective coefficients of absorption and emission at any point x along the ray. In volume rendering, the functions α and ϵ are typically reconstructed from point samples and depend upon the ray. Equations (1) and (2) provide a good approximation for a low-albedo medium [2], and are exact in the absence of scattering [3, 13].

The goal in volume rendering is to solve for $I(a,b)$ at each view ray. As this integral has no closed-form solution in general, we must resort to numerical approximation; that is, at best we can only compute the solution to within some tolerance δ. Our problem can be stated more formally as follows:

Problem ("Volume Projection"): *Given absorption and emission functions α and ϵ defined on some interval $[a,b]$ along a ray, and an error tolerance $\delta > 0$, compute an approximation $\widetilde{I}(a,b)$ to $I(a,b)$ such that $|\,\widetilde{I}(a,b) - I(a,b)\,| \leq \delta$.*

The volume projection problem is essentially one of numerical quadrature. The novelty of this particular problem lies in the special structure of the integrand and in the large computational and storage demands of rendering large datasets. Our goal, then, is to solve the volume projection problem with minimal computation by exploiting this structure. In the rest of this section, we derive an equivalent problem statement that lends itself to efficient computation, and in Section 3 we describe an algorithm for computing its solution.

2.1 Partitioning the integral

One way to make the volume rendering problem more tractable is to break up the integral in equation (1) into smaller pieces that are easier to solve [12, 14]. To make this transformation, observe first that transparency obeys a simple multiplicative rule. In particular, for any point x in the ray segment $[a, b]$, we have

$$T(a,b) \;=\; T(a,x)\, T(x,b),$$

as is easily verified from equation (2).

We can therefore discretize the ray into n contiguous segments $[d_0, d_1]$, $[d_1, d_2]$, ..., $[d_{n-1}, d_n]$ at distances $d_0 < d_1 < \cdots < d_n$ from the eye point, and rewrite equation (1) as follows:

$$\begin{aligned} I(d_0, d_n) &= \sum_{i=1}^{n} \int_{d_{i-1}}^{d_i} T(d_0, t)\, \epsilon(t)\, dt \\ &= \sum_{i=1}^{n} T(d_0, d_{i-1}) \int_{d_{i-1}}^{d_i} T(d_{i-1}, t)\, \epsilon(t)\, dt \\ &= \sum_{i=1}^{n} T(d_0, d_{i-1})\, I(d_{i-1}, d_i). \end{aligned}$$

Finally, by introducing the notation $I_k = I(d_{k-1}, d_k)$ and $T_k = T(d_{k-1}, d_k)$ we can rewrite this equation more succinctly:

$$I(d_0, d_n) \;=\; \sum_{i=1}^{n} I_i \prod_{j=1}^{i-1} T_j. \tag{3}$$

Equation (3) expresses the overall intensity of the ray in terms of the intensities and transparencies of the discrete segments. We now address the problem of approximating the intensities and transparencies of these smaller elements.

2.2 Bracketing intensity and transparency

The transparency of a segment $[d_{k-1}, d_k]$ is given by equation (2), which we repeat here using our new notation:

$$T_k \;=\; e^{-\int_{d_{k-1}}^{d_k} \alpha(s)\, ds}.$$

As we have already noted, the integral in this equation has no general closed-form solution. However, if we replace $\alpha(s)$ with an interval that bounds this function everywhere within the segment, then the integral can be written in closed form. This closed-form solution will itself be an interval that bounds the exact integral along the segment.

We therefore make the following definition. Let $\boldsymbol{\alpha}_k$ be a (nonnegative) interval $[\alpha_k^-, \alpha_k^+]$ such that $\alpha_k^- \leq \alpha(s) \leq \alpha_k^+$ for all s in $[d_{k-1}, d_k]$. We can then solve for an interval $\mathbf{T}_k = [T_k^-, T_k^+]$ bracketing the exact transparency T_k using standard interval arithmetic [10]:

$$\begin{aligned} \mathbf{T}_k &= e^{-\int_{d_{k-1}}^{d_k} \boldsymbol{\alpha}_k \, ds} \\ &= e^{-\boldsymbol{\alpha}_k \, (d_k - d_{k-1})}. \end{aligned} \tag{4}$$

Similarly, if we bracket the emission function ϵ within the segment $[d_{k-1}, d_k]$ by an interval $\boldsymbol{\epsilon}_k = [\epsilon_k^-, \epsilon_k^+]$, then it is also possible to solve for $\mathbf{I}_k = [I_k^-, I_k^+]$, an interval containing the exact intensity I_k, as follows:

$$\begin{aligned} \mathbf{I}_k &= \int_{d_{k-1}}^{d_k} e^{-\int_{d_{k-1}}^{t} \boldsymbol{\alpha}_k \, ds} \, \boldsymbol{\epsilon}_k \, dt \\ &= \begin{cases} \boldsymbol{\epsilon}_k (1 - \mathbf{T}_k)/\boldsymbol{\alpha}_k & \text{if } \alpha_k^- > 0 \\ \boldsymbol{\epsilon}_k (d_k - d_{k-1}) & \text{if } \alpha_k^+ = 0 \\ \left[\, \epsilon_k^- (1 - T_k^+)/\alpha_k^+ ,\; \epsilon_k^+ (d_k - d_{k-1}) \,\right] & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

We can then bound the overall intensity for the ray with an interval $\mathbf{I} = [I^-, I^+]$ by rewriting equation (3) with interval arithmetic:

$$\mathbf{I} = \sum_{i=1}^{n} \mathbf{I}_i \prod_{j=1}^{i-1} \mathbf{T}_j . \tag{6}$$

Finally, by introducing the notation $\|\,\mathbf{I}\,\| = I^+ - I^-$, we can rewrite the volume projection problem in terms of these smaller segments:

Problem (revisited): *Given absorption and emission functions* α *and* ϵ *defined on some interval* $[d_0, d_n]$ *along a ray, and an error tolerance* $\delta > 0$, *find a partitioning of the ray into contiguous segments* $[d_0, d_1]$, $[d_1, d_2]$, $\ldots$, $[d_{n-1}, d_n]$ *and bounds on absorption and emission* $\boldsymbol{\alpha}_k$, $\boldsymbol{\epsilon}_k$ *for each segment* $[d_{k-1}, d_k]$ *such that* $\|\,\mathbf{I}\,\| \leq \delta$, *where* $\mathbf{I}$ *is given by equation* (6).

3 Algorithm

The reformulation of the volume projection problem lends itself well to an iterative approach. The idea is to start with a single segment representing the complete intersection of the ray with the volume, and then subdivide the segment until the tolerance δ is reached.

Iterative algorithms have been described previously, for example, by Levoy [6], who refines the ray front-to-back until a certain opacity is reached. The advantage of this method is its simplicity, both in implementation and computation. However, the brute force front-to-back approach also has some drawbacks. First, the algorithm has no way of taking larger steps in places where the integrand can be easily approximated with acceptable error—for example, where it is nearly constant. Second, a strict front-to-back evaluation allows only very rough upper bounds on the error due to the remaining portion of the ray. Together, these drawbacks mean that the brute force algorithm may sometimes do more work than necessary to evaluate the integral to within a certain tolerance.

The first drawback can be addressed by incorporating some notion of hierarchy. For example, in later work, Levoy uses an octree [7], which allows the ray to pass quickly through empty regions of data.

The algorithm described here generalizes this idea further by allowing the ray to pass quickly through any region that can be integrated approximately with good error bounds—not just empty regions. Moreover, the algorithm also addresses the second drawback of a strict front-to-back approach by allowing the evaluation of the integral to proceed in an arbitrary order along the ray. In this way, the computation can always be directed toward a region of the ray that is contributing large error to the overall evaluation of the integral.

3.1 Overview

As in the octree method described by Levoy [7], we assume a hierarchical organization of the data. In our implementation, we use a BSP tree [4] instead of an octree. The hierarchical structure is view-independent, and can be built once for each dataset and stored as part of the data.

Associated with each segment k are two intervals $\mathbf{I}_k$ and $\mathbf{T}_k$, which respectively bracket the intensity and transparency of the segment. These intervals are actually three-tuples consisting of *minimum*, *estimated*, and *maximum* contributions. We denote the estimated value of an interval $\mathbf{A}$ by $\widetilde{A}$. When doing arithmetic, the minimum and maximum values are treated as ordinary intervals [10], whereas the estimated values are treated as ordinary scalars. For example, given two intervals $\mathbf{A} = [A^-, \widetilde{A}, A^+]$ and $\mathbf{B} = [B^-, \widetilde{B}, B^+]$, we have $\mathbf{A} - \mathbf{B} \;=\; [\, A^- - B^+,\ \widetilde{A} - \widetilde{B},\ A^+ - B^- \,]$.

For each ray, the algorithm maintains a set $\mathcal{S}$ of contiguous segments $\{[d_{i-1}, d_i]\}$ along the ray. Initially, the set contains just a single segment whose endpoints give the intersection of the ray with the entire volume. The algorithm proceeds iteratively, at each step *selecting* the segment from $\mathcal{S}$ that is estimated to contribute the most error, then *refining* that segment—either by splitting it in two, or else by computing its contribution more precisely—and, finally, *updating* the approximation to I based on the new $\mathcal{S}$ and associated bounds. The process is repeated until the overall tolerance δ is met.

This solution strategy is shown in pseudo code below, where V is the given volume dataset and δ is the error tolerance.

RenderVolume(V, δ)

for each ray R **do**
    let d_0, d_1 be the entry and exit points of R through V.
    initialize $\mathcal{S} \leftarrow \{[d_0, d_1]\}$.
    initialize $\mathbf{I}$ using $d_1 - d_0$ and global bounds on V.
    while $\| I \| > \delta$ **do**
        1. [*Select.*] Choose a segment $[d_{k-1}, d_k]$ from $\mathcal{S}$ to refine.
        2. [*Refine.*] Improve the bounds $\mathbf{I}_k$ and $\mathbf{T}_k$ for $[d_{k-1}, d_k]$,
            either directly or by subdividing and bounding
            the smaller segments.
        3. [*Update.*] Compute the new value of $\mathbf{I}$.
    end while
    output $\widetilde{I}$ for this ray.
end for

We now look at each of these steps—select, refine, and update—in more detail.

3.2 Selection

A good selection strategy would be to choose the segment of $\mathcal{S}$ that contributes the most error to the overall ray intensity I. Since finding these errors exactly would be equivalent to solving the original problem, we need an error *estimate* that can provide a useful heuristic for choosing the best segment to refine. Although the selection process will not be optimal, we are still guaranteed to maintain reliable bounds on I.

In principle, the error contributed by a single segment k to the overall ray intensity can be described as the sum of the error it contributes directly, and the error in its attenuation multiplied by the intensity of the combined segments behind it. We can write this error as

$$E_k \;=\; T_{\text{front}}(k) \left(\left| \widetilde{I}_k - I_k \right| \;+\; \left| \widetilde{T}_k - T_k \right| \; I_{\text{back}}(k) \right),$$

where $T_{\text{front}}(k)$ is the accumulated transparency of the segments in front of k, and $I_{\text{back}}(k)$ is the intensity of the combined segments behind k. Of course, we do not know either of these quantities exactly, but from equation (3) we can estimate them as follows:

$$\widetilde{T}_{\text{front}}(k) \;=\; \prod_{i=1}^{k-1} \widetilde{T}_i \,, \qquad \widetilde{I}_{\text{back}}(k) \;=\; \sum_{i=k+1}^{n} \widetilde{I}_i \prod_{j=k+1}^{i-1} \widetilde{T}_j \,.$$

Finally, we can bound the quantities $|\,\widetilde{I}_k - I_k\,|$ and $|\,\widetilde{T}_k - T_k\,|$ to arrive at a practical estimate $\widetilde{E}_k$ for the error contribution of segment k:

$$\widetilde{E}_k \;=\; \widetilde{T}_{\text{front}}(k) \left(\|\,\mathbf{I}_k\,\| \;+\; \|\,\mathbf{T}_k\,\| \; \widetilde{I}_{\text{back}}(k) \right).$$

Computing and updating this error estimate efficiently for each segment k is complicated by the fact that refining a single segment changes the error estimate for all the others. We therefore employ a binary tree structure, which we call a *span tree*, to aid in propagating these updates efficiently. Each leaf of the tree represents a single segment k of the ray. The intervals $\mathbf{I}_k$ and $\mathbf{T}_k$ are stored with the node. Each internal node represents the union of all the segments below it, and stores an intensity and transparency interval for this union. Using this structure, error estimates for any segment k can be computed while traversing the tree from the root to k. Note that the actual intensity I of the entire ray is always contained in the $\mathbf{I}$ interval of the span tree's root.

The selection routine descends the span tree from the root. At each node, it chooses to descend either toward the front or toward the back of the ray according to the error estimate for the node's two children:

<u>*Select* (*root*)</u>
return *RecursiveSelect* $(root, 1, 0)$

<u>*RecursiveSelect* $(s, \widetilde{T}_{\mathrm{front}}, \widetilde{I}_{\mathrm{back}})$</u>
let f be the front child of s
let b be the back child of s
$\widetilde{I}_{\mathrm{mid}} \ = \ \widetilde{I}_b + \widetilde{T}_b \widetilde{I}_{\mathrm{back}}$
$\widetilde{T}_{\mathrm{mid}} \ = \ \widetilde{T}_{\mathrm{front}} \widetilde{T}_f$
$\widetilde{E}_f \ = \ \widetilde{T}_{\mathrm{front}}(\parallel \mathbf{I}_f \parallel \ + \ \parallel \mathbf{T}_f \parallel \ \widetilde{I}_{\mathrm{mid}})$
$\widetilde{E}_b \ = \ \widetilde{T}_{\mathrm{mid}}(\parallel \mathbf{I}_b \parallel \ + \ \parallel \mathbf{T}_b \parallel \ \widetilde{I}_{\mathrm{back}})$
if $\widetilde{E}_f > \widetilde{E}_b$ **then**
    return *RecursiveSelect* $(f, \widetilde{T}_{\mathrm{front}}, \widetilde{I}_{\mathrm{mid}})$
else
    return *RecursiveSelect* $(b, \widetilde{T}_{\mathrm{mid}}, \widetilde{I}_{\mathrm{back}})$
end if

This procedure selects a "promising" leaf segment in $O(\log n)$ time. Note that the search described here does not guarantee that the leaf segment with the highest estimated error will be chosen. However, a more careful selection would require visiting all the leaf nodes. In practice, we have found that the more careful $O(n)$ search is not generally worth the extra selection cost.

3.3 Refinement

The purpose of refinement is to obtain tighter bounds on the absorption and emission within the selected segment selected, thereby improving the approximation of the overall intensity.

Refinement may be performed in different ways. For example, we may subdivide the segment and combine bounds on the two pieces to obtain a better approximation. Alternatively, we may compute the integral along the segment directly using some kind of quadrature rule, or power series expansion [11].

If refinement is done by subdivision, a splitting point must be chosen. In our implementation this point is chosen according to the splitting planes of the BSP tree. A power-series quadrature is performed whenever a leaf node of the BSP tree (i.e., a single voxel) is selected for refinement.

The quadrature rule is not strictly necessary; refinement by subdivision alone corresponds to Riemann integration, which in general has slow convergence.

3.4 Updating

Once we have refined a segment, we need to update the values of $\mathbf{I}$ and $\mathbf{T}$ in the span tree. This update is accomplished by a $O(\log n)$ bottom-up traversal of the tree, beginning with the refined span:

<u>*Update*$(s, \mathbf{T}_{\text{new}}, \mathbf{I}_{\text{new}})$</u>

[*Update the bounds associated with this leaf node.*]
$\mathbf{T}_s \leftarrow \mathbf{T}_{\text{new}}$
$\mathbf{I}_s \leftarrow \mathbf{I}_{\text{new}}$

[*Propagate the change up to the root.*]
while s has a parent **do**
  let p be the parent of s
  let f be the front child of p
  let b be the back child of p
  $\mathbf{T}_p \leftarrow \mathbf{T}_f \mathbf{T}_b$
  $\mathbf{I}_p \leftarrow \mathbf{I}_f + \mathbf{T}_f \mathbf{I}_b$
  $s \leftarrow p$
end while

Note that as a result of the update procedure, the $\mathbf{I}$ interval of the root node of the span tree contains the new estimate of the overall intensity.

3.5 Initializing the BSP tree

Our implementation uses an axis-aligned BSP tree as a simple hierarchical organization of the voxel data. This data structure can be built once and stored along with the data. Each node of the BSP tree stores upper and lower bounds for the absorption and emission coefficients α and ϵ for all the voxels represented by that node. Estimates $\widetilde{\alpha}$ and $\widetilde{\epsilon}$ are also computed by averaging the respective upper and lower bounds. This process of propagating error bounds through the hierarchical data structure is similar in spirit to the approach outlined by Laur and Hanrahan [5].

When a segment is split, the intervals $\mathbf{T}_k$ and $\mathbf{I}_k$ for a new segment k are computed from the $\boldsymbol{\alpha}$ and $\boldsymbol{\epsilon}$ values stored in the corresponding node of the BSP tree, according to equations (4) and (5). When a quadrature rule is applied, the BSP tree is queried for the values of α and ϵ at the corners of the cell. These values are used in determining the coefficients of interpolation.

4 Results

The selection and update scheme described in Section 3 has an $O(\log n)$ cost at each stage, where n is the number of segments under consideration. This computational cost is only worthwhile if it pays for itself in terms of a sufficient reduction in the overall number of refinement operations.

To evaluate this tradeoff, we implemented Levoy's octree method [7], modified to compute error bounds. In this method, the ray is refined front-to-back; thus, selection and update take constant time. The frontmost region is subdivided until it is empty, or until it is the size of a single voxel, in which case a quadrature rule is applied to machine precision. Termination occurs when the product of the accumulated transparency along the integrated portion of the ray and the maximum possible emission along the length of the unintegrated ray segment is below the error tolerance.

We tested the adaptive error bracketing scheme (AEB) described in this paper against the front-to-back method (FTB) outlined in the preceding paragraph, using three different datasets and a range of error tolerances. We measured performance in terms of both total computation time and total number of quadrature rules employed. We ran all our tests on an HP 750, a 70 MIPS machine.

Our results are shown in Figures 1, 2, and 3. For each dataset, we display a four-by-six array of images. The top three rows show images of minimum, estimated, and maximum intensity respectively. The bottom row shows the difference of maximum and minimum intensities for each pixel. The six columns show the images produced for increasingly tight error bounds. These bounds are listed below each column, both in terms of absolute error (δ) and relative error (the ratio $\delta/I_{\max}$, where $I_{\max}$ is the intensity of the brightest pixel in the highest-quality image). We also show graphs of total CPU time (in hundreds of seconds), and total number of quadratures (in thousands) below the images for each of the two methods AEB and FTB.

The first example, shown in Figure 1, is a $64 \times 64 \times 64$ dataset depicting a uniformly emissive grid behind a cloud of absorptive material that decreases linearly in density from left to right. The slow variation of cloud absorption allows for accurate bracketing by the AEB algorithm without many applications of the quadrature rule. Since the cloud is not completely opaque, the FTB algorithm cannot employ early termination, thus making it 500% to 7200% slower, as the relative error ranges from 0.2% to 42%.

The second example, shown in Figure 2, is representative of isosurface data. It is a $192 \times 112 \times 73$ CT dataset of a human pelvis. In this example, transparency drops sharply upon intersection with the bone surface, reaching negligible values after penetrating only a few voxels. In this domain, the FTB algorithm quickly locates the first intersection with the bone surface and applies the quadrature rule where it is most effective. The AEB algorithm, on the other hand, tries to find regions where the bracketed error is acceptable, but ultimately applies quadrature rules to many of the same voxels on the bone surface. On this high-gradient dataset, the extra cost of deciding where to refine causes AEB to run between 10% and 23% slower, as the relative error ranges from 0.5% to 30%.

The third example, shown in Figure 3, is a $97 \times 97 \times 116$ electron density map of an SOD enzyme. Electron density is mapped directly to absorption α, and emission ϵ is obtained via gradient shading. Absorption is low enough that rays penetrate deep into the volume, but high enough that few rays have any transparency left when they exit the dataset. For medium accuracy results (within 34%), the FTB algorithm is 52% slower than AEB. However, the benefit of using AEB gradually decreases as accuracy increases, and for the highest-precision images (within 0.5%), AEB is 23% slower even though it uses only 77% of the quadratures.

The examples suggest that adaptive bracketing is very effective in reducing the number of quadratures necessary to compute an accurate image. However, fewer quadratures does not always translate into an overall speed-up, since the extra complexity in choosing where to refine imposes additional cost. In the limit, when the highest accuracy rendering is required, there is no advantage in trying to be careful about where to refine, since any method would inevitably have to perform all of the same quadratures along the entire ray. Thus, the adaptive bracketing method pays off most when a lower-accuracy rendering is desired.

As demonstrated by the first example, the adaptive bracketing scheme has a great advantage over the front-to-back approach for “cloudy” or low-gradient data. Such datasets are best displayed with color. Although our current implementation of adaptive bracketing is limited to imaging a single wavelength at a time, the algorithm itself imposes no such restriction, and we intend to implement a color version of the algorithm soon.

5 Discussion

In this section we discuss some of the implications of adaptive error bracketing, as well as a few remaining implementation issues.

Other refinement strategies. The select–refine–update scheme, which is central to adaptive error bracketing, can make use of any tools that refine error estimates within a segment. Currently, we have implemented only two such tools: one for splitting segments, and one for refining a segment's bounds using a highly accurate quadrature. Other tools with intermediate costs and benefits might also be useful. For example, a cheaper, lower-accuracy quadrature (e.g., trapezoid rule) might be one such tool. We would also like to explore incorporating some notion of the relative costs and benefits of the various refinement tools into the selection strategy.

Appropriate error tolerances. The error bounds provided by adaptive bracketing are absolute. This type of error bound is useful if results are desired in precise units that are well-defined in advance. Unfortunately, this is rarely the case in volume rendering. Instead, once an image is created it is typically scaled to make effective use of the dynamic range of the display. This scaling also affects the absolute error. We would therefore like to investigate an iterative strategy whereby the absolute error tolerance is refined along with the image in order to achieve a prescribed “relative” error bound.

Storage requirements. One cost of the adaptive bracketing technique that we have ignored so far is the space required to store the associated data structures. The span tree takes only $O(n)$ space, where n is the linear resolution of the data, which is insignificant compared to the size of the dataset. However, the complete BSP tree, like the raw data, requires $O(n^3)$ storage. Since each node of the BSP tree currently stores about 36 bytes, this structure can become quite large. With this restriction, the largest dataset we can reasonably handle on our workstation currently has size around $n = 128$.

Adding color. As we mentioned in the previous section, color is extremely valuable in visualizing precisely those datasets for which adaptive bracketing is most effective. Color can be added in a straightforward way to our algorithm without increasing storage requirements since the BSP nodes need only bound either the greatest contribution in any one channel, or else a single luminance contribution.

6 Future work

Our initial results are encouraging and suggest a number of extensions:

Progressive refinement. Ideally, the effort invested in making a fast but low-quality image could be put to good use when making a highly-accurate rendition of the same image. The idea of progressive refinement is to generate approximate images as intermediate results on the way to the final image [1]. The progressive technique of Levoy [8] refines solutions by continually adding rays and interpolating the regions between them. An approach based on adaptive error bracketing would instead refine all rays simultaneously. However, such a scheme would require storing a span tree for every ray, which would increase the overall storage requirement to size $O(m^3 + mp^2)$, where m is the linear resolution of the volume data, and p is the linear resolution of the display. This increased storage requirement may be prohibitive for very high-resolution data or displays.

Extension to screen space. Our approach currently deals with each ray independently, as do most algorithms based on ray tracing. A promising avenue to explore is extending the algorithm to refine an entire set of rays at once, for example, the set of rays defined by a frustum penetrating the volume. In this way we might be able to take advantage of coherence among nearby rays and at the same time address the issue of antialiasing.

Bounding other errors. There are many sources of error that are currently not addressed but that may in fact be well-suited to the interval arithmetic approach described here. For example, if bounds could be placed on the data acquisition and reconstruction errors, this information could be maintained and propagated quite naturally in our scheme. The final bracketing images, then, could account for all sources of error: acquisition, reconstruction, and projection.

Irregular data. So long as we can bracket the contribution of any given sample to a segment along a ray, we can apply our technique to achieve any given accuracy. Thus, both irregularly spaced data and different filter kernels could be handled directly.

Acknowledgments

Thanks to Donald Greenberg for helpful discussions, and to Steve Westin, Dan Kartch, and Brian Smits for reviewing the manuscript. This work was supported by the NSF grant "Interactive Computer Graphics Input and Display Techniques" (CCR-8617880) and the NSF/DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219). The authors gratefully acknowledge the generous equipment grant from Hewlett Packard Corporation on whose workstations this research was conducted. The CT data was provided by the Department of Radiation Oncology at the North Carolina Memorial Hospital. The electron density dataset was provided by Duncan McRee of the Scripps Clinic and obtained as part of the Chapel Hill volume rendering test dataset.

References

[1] Larry Bergman, Henry Fuchs, and Eric Grant. Image rendering by adaptive refinement. *Computer Graphics*, 20(4):29–38, August 1986.

[2] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics*, 16(3):21–29, July 1982.

[3] S. Chandrasekar. *Radiative Transfer.* Dover Publications, New York, 1960.

[4] H. Fuchs, Z. M. Kedem, and B. F. Nayloy. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, July 1980.

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

[6] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[7] Marc Levoy. Efficient ray tracing of volume data. *Transactions on Graphics*, 9(3):245–261, July 1990.

[8] Marc Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, February 1990.

[9] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics*, 24(5):27–33, November 1990.

[10] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, 1966.

[11] Kevin L. Novins and James R. Arvo. A power series algorithm for highly accurate volume rendering. Submitted to the *1992 Boston Workshop on Volume Visualization.*

[12] Paolo Sabella. A rendering algorithm for visualizing 3d scalar fields. *Computer Graphics*, 22(4):51–57, August 1988.

[13] V. V. Sobolev. *A Treatise on Radiative Transfer.* D. Van Nostrand Company, Princeton, New Jersey, 1963. Translated by S.I. Gaposchkin.

[14] Craig Upson and Michael Keeler. V-buffer: Visible volume rendering. *Computer Graphics*, 22(4):59–64, August 1988.

[15] Lee Westover. Interactive volume rendering. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, May 1989.

[16] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

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

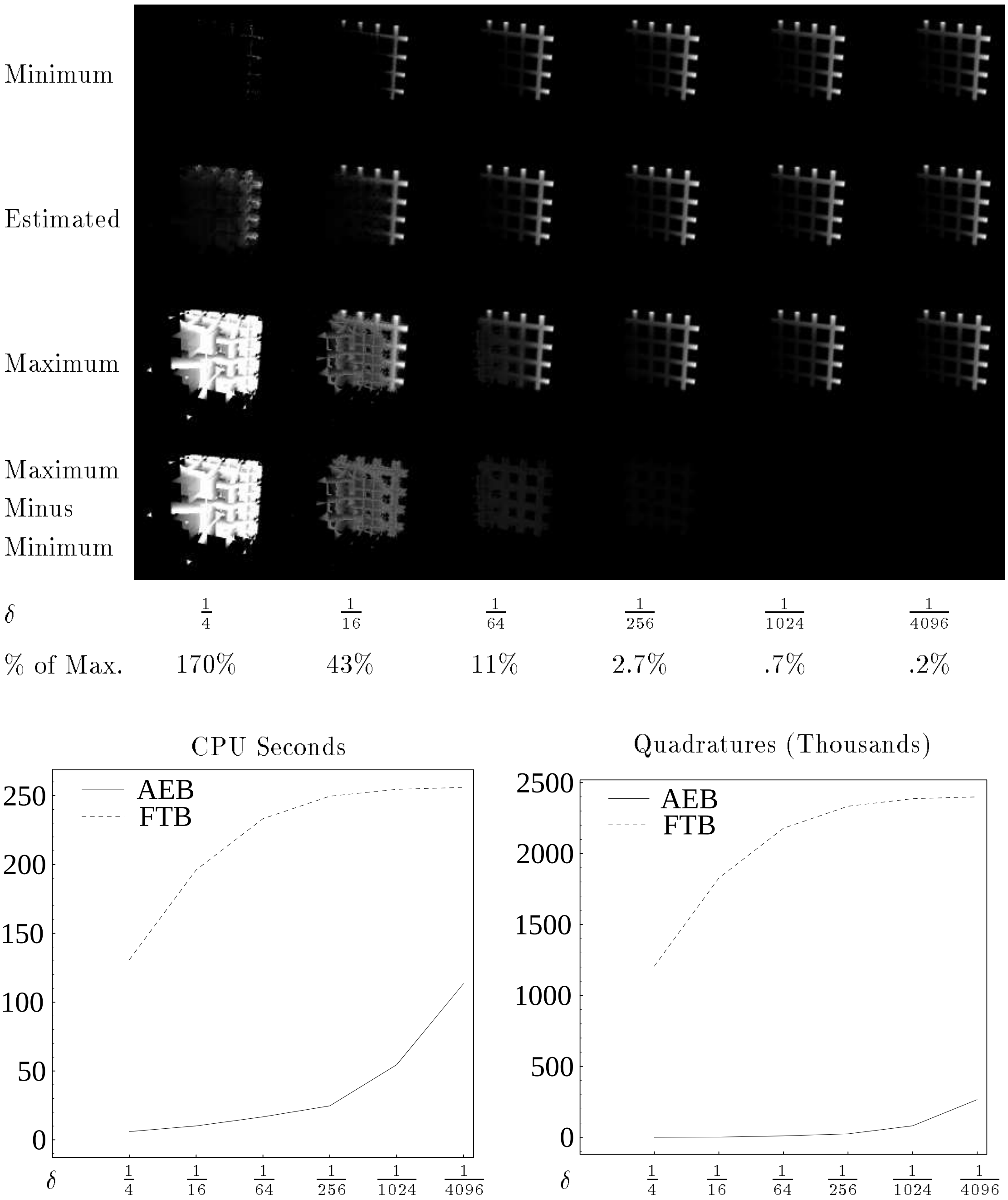

Figure 1 Emitting grid behind absorbing cloud of linearly varying density.

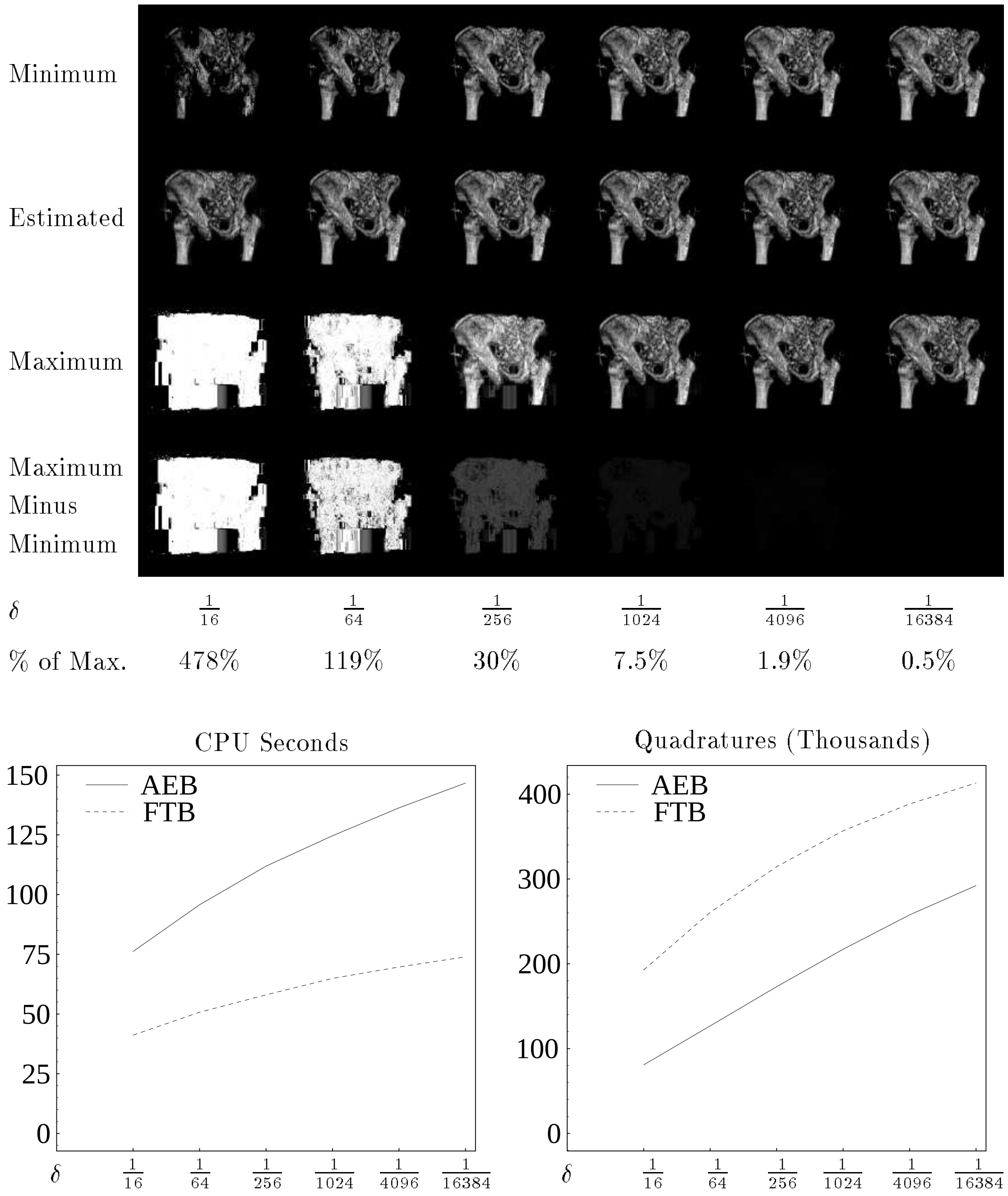

Figure 2 Bone iso-surface with gradient shading.

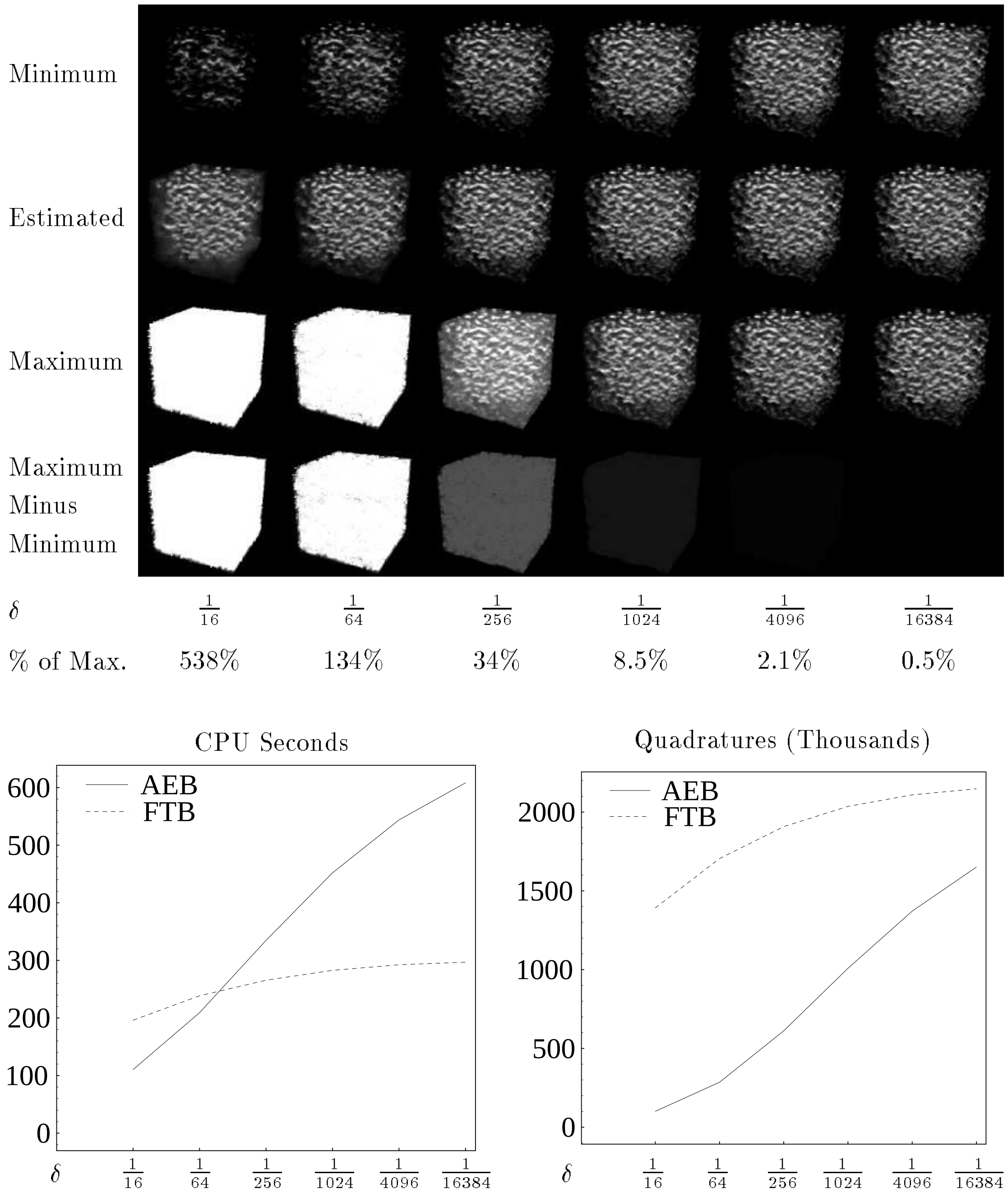

Figure 3 Electron density of the SOD enzyme.