Fast Algorithms for Volume Ray Tracing

John Danskin and Pat Hanrahan

Abstract

We examine various simple algorithms that exploit homogeneity and accumulated opacity for tracing rays through shaded volumes. Most of these methods have error criteria which allow them to trade quality for speed. The time vs. quality tradeoff for these adaptive methods is compared to fixed step multiresolution methods. These methods are also useful for general light transport in volumes.

1 Introduction

We are interested in speeding volume ray tracing computations. We concentrate on the one dimensional problem of tracing a single ray, or computing the intensity at a point from a single direction. In addition to being the kernel of a simple volume ray tracer, this computation can be used to generate shadow volumes and as an element in more general light transport problems. Our data structures will be view independent to speed the production of animations of preshaded volumes and interactive viewing.

In [11] Levoy introduced two key concepts which we will be expanding on: *presence acceleration*, and α-*Termination*. The first technique allows fast traversal of empty space using octrees. The second technique terminates front to back ray-tracing after the opacity accumulated by the ray exceeds a certain threshold.

We will expand on these two basic ideas:

1. We exploit homogeneity in the volume dataset, not just presence.
2. We gradually take fewer samples and reduce the precision of our calculation as a ray accumulates opacity in front to back ray-tracing.

A general motivation for studying these techniques is that they are continuous, and do not involve binary valued data-structures or all or nothing decisions. We will relate these ideas to importance sampling [4]: a fundamental technique in Monte Carlo integration.

John Danskin jmd@cs.princeton.edu
Pat Hanrahan pmh@cs.princeton.edu
Department of Computer Science
Princeton University
Princeton, NJ 08544-2087

In Section 2 we will discuss the light transport equations which we will be analyzing. In Section 3, we will discuss importance sampling and its relationship to light transport. In Section 4, we will preview the proposed acceleration algorithms. In Section 5, we will describe our experiments. In Section 6, we will discuss our data-structures. In Section 7, we will describe our algorithms in more detail. In Section 8, we will present our results. Finally, in Section 9, we will discuss some of the implications of our algorithms and results.

2 Light Transport

The shaded volume is treated as a mass of glowing jello where the color and density of the jello are sampled along a regular three-dimensional grid. Color and density values at intermediate points are usually derived using tri-linear interpolation [11], although higher order interpolation is possible. We limit the gamut of color and density to the range $[0-1]$ before pre-multiplication, and store color values premultiplied by the density.

To render this volume, we need to find the intensity and color of the light that would impinge on a given spot of film in the back of a pinhole camera (for instance). The pinhole geometry naturally gives rise to the idea of ray tracing. The origin of the ray is the center of the differential area of film to be exposed. The direction of the ray is towards the pinhole in the camera. The value of the ray is the integral of all of the light found along the ray, scaled by the optical distance from the light to the film.

$$I(\vec{x},\vec{\omega}) = \int_0^T e^{-\int_0^t \sigma(s)ds} I(t)dt \tag{1}$$

Where $\vec{x}$ is the origin of the ray, $\vec{\omega}$ is the unit direction vector of the ray, $\sigma(s)$ is the differential attenuation at $\vec{x}+s\vec{\omega}$, and $I(t)$ is the differential intensity scattered at $\vec{x}+t\vec{\omega}$ in the direction $-\vec{\omega}$.

The most straightforward volume ray tracing technique is to turn the nested integral of equation 1 into a nested summation with equally spaced samples. Here the sample spacing is assumed to be unity.

$$I = \sum_{0\le i\le n} \left(\prod_{0\le j\le i} e^{-\int_j^{j+1}\sigma(s)ds} \right) I(i) \tag{2}$$

if we define

$$\alpha(j) = 1 - e^{-\int_j^{j+1}\sigma(s)ds} \tag{3}$$

and

$$\beta(j) = (1 - \prod_{0\le j\le i} 1-\alpha(j)) \tag{4}$$

we have

$$I = \sum_{1 \le i \le n} (1 - \beta(i))I(i) \tag{5}$$

where $\alpha(j)$ is the fraction of light which is stopped by the material at j. $\beta(i)$ (the accumulated opacity at i) is the fraction of light which is stopped by the material from 0 to i.

With this insight, we see that in the special case where $n = 2$, this sum embodies the Porter-Duff *over* operator [20].

```
rgba
over(rgba fg, bg) {
    return fg + (1 - fg.alpha) * bg;
}
```

the resulting alpha can be seen to be

$$(1 - (1 - fg.alpha)(1 - bg.alpha)) \tag{6}$$

which fits equation 5. Successive applications of the *over* function solve equation 5 with $O(n)$ operations instead of the $O(n^2)$ operations which result if equation 5 is programmed naively. The sum in equation 5 is equivalent to the nested integral in equation 1 in the limit as the sample spacing goes to zero [21; 17]. We will be concerned with efficiently approximating this sum.

There has been a lot of work on efficiently and accurately rendering volumes with large or irregularly shaped voxels. Max et. al. [17] discuss rendering 3D scalar fields sampled on the vertices of a space filling arrangement of convex polyhedra. Upson and Keeler [23] discuss rendering volumes with large voxels. Garrity [6], Wilhelms et. al. [25], Shirley and Tuchman [22], and Neeman [18] discuss rendering irregular volumes and volumes specified on curvilinear grids. We will assume that our input volume has sufficiently fine resolution so that the approximation of equation 5 is sufficiently accurate, and also assume that our volume dataset is defined on a regular grid.

Here is the basic volume ray tracing pseudo code which we will be modifying to implement our various acceleration techniques:

```
rgba Trace(volume, O, D, t, maxt) {
  rgba c, s;
  c = (0,0,0,0)
  while (t < maxt) {
    s = sample(volume, O + t D);
    c = over(c, s);
    t++;
  }
  return c;
}
```

3 Importance Sampling

The basis for most of our acceleration techniques is importance sampling [4]. Importance sampling applied to integrals means concentrating most of the samples in that portion of the domain which contributes most to the value [3].

We will follow [3] in the derivation of equations 7 through 10. Suppose we need to determine the value of $f(x)$ where x is sampled from a probability distribution $p(x)$ which is defined on $[a, b]$.

$$E(f) = \int_a^b f(x)p(x)dx \tag{7}$$

If we need to sample x from some other probability distribution $p^*(x)$ instead, we can compensate by introducing $w(x_i) = p(x_i)/p^*(x_i)$ and $f^*(x) = w(x)f(x)$.

$$E(f^*) = \int_a^b w(x)f(x)p^*(x)dx = E(f) \tag{8}$$

The expected value of the function is unchanged, but the variance is different:

$$\begin{aligned} V(f^*) &= E(f^{*2}) - E^2(f^*) \\ &= \int_a^b \frac{p^2(x)}{p^{*2}(x)} f^2(x)p^*(x)dx - E^2(f) \\ &= \int_a^b \frac{p(x)}{p^*(x)} f^2(x)p(x)dx - E^2(f) \end{aligned} \tag{9}$$

Intuitively, this means that we can reduce the variance by concentrating samples where $f^2(x)p(x)$ is relatively large. If $p^*(x) = f(x)p(x)/E(f)$ then $V(f) = 0$ since then

$$E(f^{*2}) - E^2(f) = \int_a^b \left(f(x)\frac{p(x)}{p^*(x)} - E(f) \right)^2 p^*(x)dx = 0 \tag{10}$$

Of course this is not immediately useful because we would not be considering sampling the integral if we knew $E(f)$. However, consider equation 1.

$$I(\vec{x}, \vec{\omega}) = \int_0^T e^{-\int_0^t \sigma(s)ds} I(t)dt$$

If we divide both sides of the equation by the constant

$$C = \int_0^T e^{-\int_0^t \sigma(s)ds} dt \tag{11}$$

then we can set

$$p(t) = \frac{e^{-\int_0^t \sigma(s)ds}}{C} \tag{12}$$

since $p(t)$ is positive and sums to one, it is a probability distribution function, and we have

$$\frac{I(\vec{x}, \vec{\omega})}{C} = \int_0^T p(t)I(t)dt \tag{13}$$

Since this integral differs only by a constant from the integral in equation 1, the ideal sampling pattern for this integral will also be the ideal sampling pattern for equation 1. Plugging in equation 3, the ideal sampling distribution is

$$p^*(t) = \frac{I(t)p(t)}{I(\vec{x}, \vec{\omega})} \tag{14}$$

Of course, we don't know what the value of $I(\vec{x}, \vec{\omega})$ is ahead of time, and $I(t)$ and $p(t)$ are only available by sampling, but we can make some qualitative statements about good sampling technique:

- Sample density should vary linearly with $I(t)$.
- Sample density should vary linearly with $p(t) = \frac{1-\beta(j)}{C}$.
- Sample weights should vary inversely with sample density.
- Sample density calculations are complicated by the need to calculate the integral $p(t)$ accurately, but observe that $p(t)$'s ideal sample distribution is $p(t)$ (within a constant). Since we calculate both integrals together in front to back order, we have a good estimate of $p(t)$ when we need it.

4 Overview of Algorithms

We will discuss the following techniques:

1. *Fixed Step Multiresolution*: We maintain an *average* pyramid of volumes where each successive volume is half the resolution of the previous one. The level in the pyramid at which the algorithm operates is user specified. A more sophisticated variant of this algorithm is due to Levoy [13]. We use this algorithm as a control for comparison with our other algorithms.
2. *Presence-acceleration* uses a $max_{27}(\alpha)$ pyramid (defined below) to maintain the maximum material density in a neighborhood, and an *average* pyramid to maintain average material properties. Samples are taken at higher levels of the pyramid in regions of low density.
3. *Homogeneity-acceleration* uses a $range_{27}$ pyramid (defined below) to maintain a measure of local homogeneity, and an *average* pyramid to maintain average material properties. Samples are taken at higher levels of the pyramid in regions of high homogeneity [10].
4. *Russian Roulette* [9] [1] [16] [3] probabilistically kills off some of the front to back ray-tracing calculations once they have accumulated opacity exceeding some threshold. Surviving calculations are increased in weight so that the average value returned is unbiased.
5. *β-Acceleration* allows the ray calculation to sample higher in a pyramid as the optical distance (β) increases. The idea is that as less and less of the sampled light reaches the eye, errors in the amount of light sampled become less important.

5 The Experiment

We have conducted experiments with a number of plausible algorithms, separately and combined (Figures 3-5). All of the algorithms described below allow a time vs. accuracy tradeoff, and so we present plots of image error vs. number of samples and image error vs. time for each method. Finding a good metric for image error is a difficult task, with non-linear perceptual differences abounding. We use the average three dimensional Manhattan distance between corresponding pixels of an image, scaled from 0 to 1 as our metric.

$$NYNYdist(a,b) = (|a.r - b.r| + |a.g - b.g| + |a.b - b.b| + |a.a - b.a|)/4 \quad (15)$$

An average image error of 20% seems to be the maximum useful error limit. All timings represent user time on an SGI Power Series 220.

So that readers can make their own decisions about error tolerances, and so that readers can see what the errors made by a particular method look like, we have included a table of images with the graphs. Each entry in the table consists of three images: the ray traced image, its *diff* with a reference image, and an image representing the number of samples necessary to compute each pixel. The sample images all have the same scale, so it is possible to compare sample images from different algorithms, but they have not been gamma corrected for printing, so quantitative comparisons of particular pixels are not possible.

Images read (in order of increasing error) from left to right. Pictures are on top, then diffs, and finally number of samples.

6 Pyramid Data Structures

Volume pyramids are represented as a set of volumes. The first level (level 0) of the pyramid is the original data. The second level of the pyramid has about one eighth as many entries as the first level. Because the original data is not necessarily a cube with power of 2 sides, we set the size of the i-th dimension of level n to:

$$size[n][i] = max((size[n-1][i]+1)/2, 1) \quad (16)$$

until we have a level with dimensions $1 \times 1 \times 1$, which is the top. Since some of the children of an entry in the nth level may be outside of the domain of the original dataset, we define the dataset to take on a constant value outside of its domain. This is a compact representation for a pointerless complete octree [24] allowing efficient neighbor, parent and child calculations.

In the *average* pyramid, each sample in the nth level of the pyramid stores the average of it's eight children in the $n-1$st level of the pyramid. Since we store our data in an 8 bit per channel format, we find it necessary to dither when averaging down to maintain overall average node values. A voxel in the *average* pyramid is the cubic region supported by 8 data values. To sample within the voxel we apply trilinear interpolation [11] (although higher order interpolation is possible).

Levoy used *average* pyramids for gaze directed rendering [13]. They are a 3D generalization of *mip-maps* [26].

When ray tracing with the *average* pyramid, samples can be taken at any level. When we sample at level 1 of the pyramid, that means that we are taking a step of size 2, so the result of the sample should be the same as if we had taken 2 steps of size 1 at level 0 (assuming that the samples values at level 0 are the same as the sampled value at level 2). To achieve this, we apply the following filter:

```
rgba ScaleColor(int n, rgba ia) {
    while (n--) {
        ia = over(ia, ia);
    }
    return ia;
}
```

This filter is the same as the one proposed in [10].

We tried storing these scaled values in the pyramid instead of the averages, but the quantization effects were intolerable at eight bits per entry. At this point, we use *ScaleColor* as a filter for volume access. First trilinear interpolation is applied, then we scale colors and opacity. A better solution might be to store levels $n > 0$ of the pyramid in higher precision.

In implementing some form of *presence-acceleration*, where we want to step quickly over areas of low density, we need a data structure which allows access to the maximum densities we might run into in such a big step. Wilhelms and Van Gelder propose storing child min and max values in octree nodes to speed iso-surface generation [24]. At first glance, a *max* pyramid looks like the right structure for us. Unfortunately the *max* pyramid doesn't efficiently solve our problem because a ray taking even a tiny step can step through three pyramid nodes. The max_{27} pyramid replaces each level 0 voxel in the *average* pyramid with the maximum value in the voxel and its 26 neighbors. Since we are using trilinear interpolation in the *average* pyramid, this means that each *max* voxel is the maximum of the 64 data points supporting the 27 voxels in the neighborhood. This calculation can be eased by noticing that adjacent *max* voxels share 48 data points, so the incremental work needed per *max* voxel is a little more than computing the *max* of 16 data points. Higher levels of the max_{27} pyramid contain the same information, but at half the resolution: a second level max_{27} pyramid voxel is supported by $27 \times 2 \times 2 \times 2 = 216$ voxels and 343 data points. Luckily, this data is summarized in 8 voxels in the first level of the pyramid. See Figure 1 for an example of how this works in 2D.

Since we are storing worst case information about a neighborhood, interpolation in the max_{27} pyramid is inappropriate, and a voxel is supported by a single data point (considered to be located in the center of the voxel).

The max_{27} pyramid is useful in its own right; however we will use it and the analogous min_{27} pyramid to compute

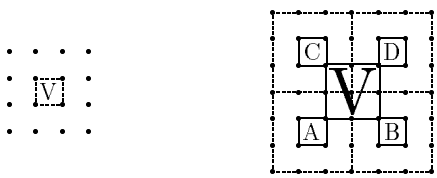

Figure 1: Left: 2D neighborhood of voxel V at level 0. Right: 2D neighborhood of Voxel V at level n, showing relevant neighborhoods ABCD in level $n-1$. Since the neighborhoods ABCD span the neighborhood V, the *max* of neighborhood V is just the *max* of neighborhoods ABCD.

another pyramid, the $range_{27}$ pyramid. Each voxel in the $range_{27}$ pyramid contains the Manhattan distance (equation 15) between the scaled RGBα value derived from sampling the min_{27} pyramid for each channel and the scaled value derived from sampling the max_{27} pyramid for each channel. Intuitively, whenever this range is small, the region is nearly homogeneous, and it is safe to take a big step using an average value for the region. Whenever the range is large, the region is heterogeneous, and small steps are necessary. Pseudo code to compute the $range_{27}$ map follows:

```
minV = computeMinVol(v);
maxV = computeMaxVol(v);

forEach(level in v) {
  l = level;
  forEach(pt in v) {
    errorV(level, pt) =
      NYNYdist(
        ScaleColor(l, maxV(l, pt)),
        ScaleColor(l, minV(l, pt)));
  }
}
```

The $range_{27}$ pyramid will be our primary acceleration data structure, but we could have used a $variance_{27}$ pyramid instead [10]. We chose the $range_{27}$ pyramid because it has good worst case behavior. A large voxel high up in a pyramid could be mostly empty, containing only a very thin yet totally opaque wall, and yet have low variance because of the small number of variant pixels. This could lead to dramatic image artifacts. The $range_{27}$ pyramid cannot make this kind of mistake because it encapsulates the extremes of the local data and not just the average difference from the mean.

These pyramid datastructures are view independent: the cost of construction can be amortized over a whole animation sequence or interactive session.

7 Algorithms

7.1 Fixed Step Multiresolution

Levoy [13] has proposed using an *average* pyramid or mipmap [26] to accelerate the tracing of rays that are intended to cover a large image area. The density of rays per unit area, and the number of samples per ray are both decreased to produce fuzzy pictures very fast. The intended use for this technique is gaze directed rendering. By interpolating between adjacent levels of the pyramid when sampling, smooth variations in the level of detail can be accomplished.

We have implemented a variant of this technique as a control for more complicated acceleration techniques. The active resolution level is set by the user, and the ray tracer generates an entire picture at that resolution. Inter-ray spacing is unaffected by the resolution level, but the inter-sample spacing within a ray is set to $pow(2, level)$. The *fixed step multiresolution* method at full resolution was used to generate our reference image. The *diff* images in the figures represent the pixel and channel-wise absolute value of the difference between the test image and the reference image.

Note that for comparison purposes with Levoy's implementation in [13] our implementation uses almost exactly half as many samples per ray because we do not implement interpolation between adjacent pyramid levels.

7.2 Presence-Acceleration

We saw in Section 3 that it is not necessary to take many samples in regions with small contributions to the opacity and intensity integrals. *Presence-acceleration* takes advantage of this by using average values to represent large nearly empty sample domains. We determine that a sample domain must have a small contribution to the integral by examining a $\max_{27}(\alpha)$ pyramid in the locale of interest. If the maximum α value in the region is less than some user specified threshold k, the region is approximated with a single sample: the appropriately weighted value sampled from the *average* pyramid. Recall that we use the *ScaleColor* filter to achieve proper weighting. In the special case where $k = 0$, this method is equivalent to Levoy's method for traversing empty space without error [11].

The following pseudo code implements *presence acceleration*:

```
rgba Trace(pyramid, max_alpha, k, O, D, t, maxt) {
      rgba c, s;
      int level = 0;

      c = (0,0,0,0);
      while (t < maxt) {
        if ((level == 0) ||
[1]     (sample(max_alpha, level, O + t D) <= k)) {
          s = sample(pyramid, level, O + t D);
          c = over(c, s);
          t += pow(2, level++);
        } else {
          level--;
        }
      }
      return c;
    }
```

If $(level > 0)$ then the sample on line [1] must be less than or equal to k in order to make progress. Whenever we can make progress, we try to move up the tree. Whenever we cannot make progress, we move down the tree. We can always make progress at level 0.

7.3 Homogeneity-Acceleration

From the point of view of importance sampling, *presence-acceleration* makes two mistakes:

1. It ignores accumulated opacity
2. It moves up the pyramid too fast. In importance sampling, it is desirable to get about the same amount of "stuff" under each sample. *Presence-acceleration* will use a single sample for an arbitrarily large region if the maximum value in that region is less than k.

Homogeneity-acceleration addresses the second problem, and also incorporates an optimization outside of importance sampling. If there is a region where the value of a function is known, then it isn't necessary to sample it. In area where the volume is homogeneous, a single sample from the *average* pyramid can provide a good approximation to the region of

homogeneity, regardless of how much "stuff" is in the region. The $range_{27}$ pyramid defined in Section 6 provides a good heuristic for deciding whether an average value is a good fit to the function. Not only is the size of the range of volume values encapsulated in the $range_{27}$ pyramid, but since the volume values are scaled before the difference is generated, they represent over(under) estimates of the amount of integrable "stuff" in the region of interest. The scaling process even correctly accounts for the increase in accumulated opacity across the region, thus these estimates will diverge appropriately as we move up the pyramid, and the $range_{27}$ pyramid will limit inappropriate acceleration.

The range heuristic is not perfect in that errors in α affect the rest of the ray computation, while errors in RGB are independent, so α should be more important. How much more important depends on what follows in the computation, but we don't know anything about the rest of the computation, because we are proceeding from front to back.

The implementation of *homogeneity-acceleration* is exactly the same as the implementation of *presence-acceleration* except that the $max_{27}(\alpha)$ pyramid is replaced everywhere with the $range_{27}$ pyramid.

7.4 Accumulated Opacity Algorithms

As we trace a ray through a volume from front to back, we accumulate opacity. For instance, if we have an accumulated $\beta = .6$, only 40% of the light we find will make it back to the eye. This means that any mistakes we make will be similarly scaled. Hall implemented a weight cutoff mechanism for a ray tracer [7] and Levoy adapted this technique to volume ray tracing: any rays which accumulated an α larger than some threshold were terminated. Unfortunately this leads to a systematic bias in the image [1]. We will develop or adapt two techniques to take advantage of accumulated opacity without introducing a systematic bias: Russian Roulette [9], and *β-acceleration*.

7.4.1 Russian Roulette

Russian Roulette was developed as a way for particle transport codes to avoid simulating particles with low weights without biasing the results. The algorithm reported in [1] uses Russian Roulette to cut down on the bushiness of shade trees in classic ray tracers. We use Russian Roulette to cut down on the average penetration of rays into the volume.

Think of the ray as a particle. Every time it passes through a voxel, there is some probability that the particle was scattered by some of the material in the voxel. This probability $p(t)$ is equal to $\alpha(t)$ at the voxel. We could end the calculation at t with probability $p(t)$ reporting $I(t)$ as the color of the particle, but this would lead to a high variance in reported values. Instead, we typically modify the color of the particle according to the current weight and decrease the weight of the particle according to $p(t)$. (This is the *over* operation. Decreasing the weight of the particle is accomplished by increasing the accumulated β.) This continuous calculation is correct, but it does the same amount of work behind the opaque wall as in front of it. What we want is a method to eliminate some of the computation without biasing the result. In Russian Roulette, we probabilistically terminate some of the rays which have low weight, but increase the weight (and thus the amount of light reported by) the survivors so that the total amount of light found in the whole image is not biased. We are also careful to ensure that the average weight of rays achieving a given β (accumulated opacity) is also unbiased.

We note that Russian Roulette is a special case of importance sampling which takes into account accumulated opacity $\beta(t)$, but not local luminance $I(t)$.

The pseudo-code below implements Russian Roulette:

```
rgba Trace(volume, O, D, t, maxt) {
  rgba c;
  rgba s;
  float weight = 1.0, W;

  c = sample(volume, O + t D);
  while (++t < maxt) {
    W = weight * (1 - c.alpha);
    if (W < Thresh) {
      Pdie = 1 - W / Thresh;
      choose x from [0, 1);
      if (Pdie > x) {
        break;
      } else {
        weight *= 1/(1 - Pdie);
      }
    }
    s = sample(volume, O + t D);
    c = over(c, scaleRgb(s, weight));
/* note: only RGB are scaled */
  }
  return c;
}
```

7.4.2 β-Acceleration

β-acceleration adds the ability to accelerate after accumulating opacity to *homogeneity-acceleration*. Referring back to Section 3, we see that sample density should vary inversely with increasing opacity. Intuitively, *homogeneity-acceleration* takes weighted samples, in which each sample has some bounded error. Before these samples are incorporated into the intensity integral, they are attenuated by the optical distance along the ray. When the sample value is attenuated, the error in the sample value is attenuated too. It is reasonable then to attenuate the error estimate sampled from the $range_{27}$ pyramid before using it.

Note (comparing the tables in Figures 3 and 5) that for a given value of k, *β-acceleration* will usually generate an image with more error than *homogeneity-acceleration* because *β-acceleration* is realistic about the effects of errors after opacity is accumulated, while *homogeneity-acceleration* is relatively paranoid.

We modify line [1] of the code for *homogeneity-acceleration* to implement *β-acceleration*:

```
    rgba Trace(pyramid, range, k, O, D, t, maxt) {
      rgba c, s;
      int level = 0;

      c = (0,0,0,0);
      while (t < maxt) {
        if ((level == 0) ||
[1]         (1 - co.a) *
            (sample(range, level, O + t D) <= k)) {
          s = sample(pyramid, level, O + t D);
          c = over(c, s);
          t += pow(2, level++);
        } else {
          level--;
        }
      }
      return c;
    }
```

Figure 2 has a graph showing the history of a ray traced using *β-acceleration*. Notice how the ray skips quickly across empty space until it hits the lobster's claw, where it samples carefully until accumulated opacity builds up. Then the ray accelerates exponentially through the rest of the volume (by moving up through the pyramid), approximating the rest of the volume including the lobster's other claw with only four samples. The ability to produce these graphs interactively during rendering was a great debugging aid.

8 Results

All of our experiments were run on the same test data (shown in Figure 7). The cylinder head data set was classified and shaded to include a few very opaque regions, a lot of homogeneous nearly transparent regions, and hardly any empty space (although there is some). Although *homogeneity-acceleration* can skip over empty space as well as *presence-acceleration*, it needs a fuzzy dataset like this one to significantly outperform *presence-acceleration*.

Figure 3 shows that *homogeneity-acceleration* and *presence acceleration* are about 30% faster than vanilla volume ray tracing for this dataset when constrained to make no error. *Homogeneity-acceleration* is able to degrade image quality smoothly with increasing k, while *presence-acceleration* is bimodal, making no error at all, and then suddenly jumping to 10% error. *Presence-acceleration* makes this sudden jump because the dataset has large regions of low opacity. As soon as k is large enough to allow *Presence-acceleration* to move above level 0 anywhere, it moves quite high in the pyramid. *Homogeneity-acceleration* degrades more gradually because (as mentioned above) it's *range* values are scaled to take into account the size of the region sampled. The acceleration methods generally do better with the number of samples, than they do with the number of seconds, when compared to the *Fixed Step Multiresolution* method, because of the extra overhead necessary to find the right level for stepping. Note that if you are willing to put up with 5% image error, *homogeneity-acceleration* runs about 3 times faster than the vanilla ray tracer (taking less than one sixth as many samples), and more than twice as fast as the best run of *presence-acceleration* which has less than twice the error (k is evenly spaced in both cases for a fair comparison).

In Figure 4 we can see that Russian roulette does about 10% better than no Russian Roulette as long as the threshold variable is kept high enough so that the added error is low (above about .9), but lowering the threshold variable further doesn't speed calculations much, while adding noise to the resulting picture. This result holds for both vanilla Russian Roulette and Russian Roulette combined with *homogeneity-acceleration*.

Figure 5 demonstrates that *β-acceleration* clearly dominates Russian Roulette, and is at least a 10% improvement over simple *homogeneity-acceleration*. Clearly, given a *homogeneity-acceleration* implementation, *β-acceleration* is worth the eight extra keystrokes that it takes to implement. We summarize the performance of *β-acceleration* compared to the previous state of the art in Figure 6. If we are willing to accept an image error of 5%, *β-acceleration* outperforms the empty space skipping algorithm in [11] (including *α-Termination*) by a factor of 2.74 in this experiment.

9 Discussion

We have extended the volume homogeneity ideas of Laur [10], and the volume presence ideas of Levoy [11], by combining them with importance sampling. The resulting algorithm is simple and has markedly improved performance over previous volume ray tracers, while retaining the intrinsic image quality advantages of ray tracing as compared to splatting.

Although *β-acceleration* is fast, it is hardly fast enough for real time rotation, even with the addition of progressive and adaptive refinement [2; 15]. Splatting [10] can support real time rotation, but image quality suffers even at peak resolution. We expect that it should be possible to construct a volume beam tracer which combines the image quality of ray tracing with the performance of splatting.

Since the quality of images produced with *β-acceleration* degrades smoothly with increasing values of k, it would be reasonable to use k as an interactive or automatic control for adjusting the speed vs. image quality tradeoff. k could be tied to the rotation rate for example, so that images could be generated very quickly while the volume is moving. In the *Fixed Step Multiresolution* method, the user picks the number of samples (or speed), and gets an image with some error. In *β-acceleration* the user specifies an error, and the algorithm picks the number of samples.

We would like to see importance sampling combined with stratified sampling [8; 16] (also known as quota sampling) to justify and perhaps refine the commonly used technique of adaptive refinement [2; 19; 15]. In stratified sampling, the sample space is broken up into disjoint regions, and a fixed number of samples are generated in each region. A new sample distribution is generated according to the sampled variance of each region, and the sampling process is repeated until all of the regions achieve an acceptable sampled variance.

Further research in volume rendering and global illumination of volumes will benefit from a careful study of the existing Monte Carlo literature as the problem of tracking light in a semi-transparent volume is very closely related to the problem of tracking particles.

10 Acknowledgments

Thanks to Larry Aupperle for thinking of *β-acceleration*, pointing us in the right direction, and otherwise making this work possible. David Laur contributed ideas, structure, and a superb graphics environment. We thank Silicon Graphics Incorporated for a generous equipment grant. This research was funded in part by the National Information Display Laboratory at David Sarnoff Research Center.

References

[1] Arvo, James and David Kirk "Particle Transport and Image Synthesis" *Computer Graphics* Vol. 24, No. 4, August 1990

[2] Bergman L., H. Fuchs, E. Grant, and S. Spach "Image rendering by adaptive refinement," *Computer Graphics*, Vol. 20, No. 4, August 1986, pp. 29-37

[3] Carter, L. L. and E. D. Cashwell "Particle-Transport Simulation with the Monte Carlo Method" *Technical Information Center, Energy Research and Development Administration 1975* ISBN 0-87079-021-8

[4] Coveyou, R. R., V. R. Cain, and K. J. Yost, "Adjoint and importance in Monte Carlo application" Nuclear Science Engineering, No. 27, pp. 219-234 (1967)

[5] Drebin, R.A., L., Carpenter and P. Hanrahan, "Volume Rendering" Computer Graphics, Vol. 22, No. 4, August 1988, pp. 65-74

[6] Garrity, M., "Ray Tracing Irregular Volume Data," Computer Graphics, Vol. 24, No. 5, November 1990

[7] Hall, R., and D. Greenberg "A testbed for realistic image synthesis," *IEEE Computer Graphics and Applications*, Vol. 3, No. 10, November 1983, pp. 10-20.

[8] Hammersly, J. M., and D. C. Handscomb, "Monte Carlo Methods," in *Methuen's Monographs on Applied Probability and Statistics*, Methuen and Company, Ltd., London, 1964.

[9] Kahn H., "Use of Different Monte Carlo Sampling Techniques" Symposium on Monte Carlo Methods, University of Florida, March 1954 *Credits J. von Neumann and S. Ulam with the idea of Russian Roulette*

[10] Laur, D., and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering" *Computer Graphics* Vol. 25, No. 4, July 1991, pp. 285-288

[11] Levoy, M., "Efficient Ray tracing of Volume Data" *ACM Transactions on Graphics,* Vol. 9, No. 3, July 1990

[12] Levoy, M., "Volume Rendering by Adaptive Refinement" *The Visual Computer,* Vol. 6, No. 1, February 1990, pp. 2-7

[13] Levoy, M., "Gaze-Directed Volume Rendering" *Computer Graphics* Vol. 24, No. 2, March 1990

[14] Levoy, M., "Display of Surface from Volume Data" *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May 1988, pp. 29-37

[15] Levoy, M., "Volume Rendering by Adaptive Refinement," *The Visual Computer*, Vol. 6, No. 1, February 1990, pp. 2-7

[16] Lux, I., and L., Koblinger "Monte Carlo Particle Transport Methods: Neutron and Photon Calculations" *CRC Press 1990* ISBN 0-8493-6074-9

[17] Max, N., P. Hanrahan, and P. Crawfis, "Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions" Computer Graphics, Vol. 24, No. 5, November 1990

[18] Neeman, H., "A Decomposition Algorithm for Visualizing Irregular Grids" Computer Graphics, Vol. 24, No. 5, November 1990

[19] Painter, J., and K. Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement," *Computer Graphics*, Vol. 23, No. 3, July 1989, pp 281-288.

[20] Porter T., and T. Duff "Compositing Digital Images" *Computer Graphics*, Vol 18, No. 3, July 1984, pp 253-259

[21] Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar Fields" *Computer Graphics* Vol. 22, No. 4, August 1988, pp. 51-58

[22] Shirley, P., and A. Tuchman, "A Polygonal Approach to Direct Scalar Volume Rendering" Computer Graphics, Vol. 24, No. 5, November 1990

[23] Upson C., M. Keeler, "V-BUFFER: Visible Volume Rendering" Computer Graphics, Vol. 22, No. 4, August 1988, pp. 59-64

[24] Wilhelms, J. and A. Van Gelder "Octrees for Faster Isosurface Generation," Computer Graphics, Vol. 24, No. 5, November 1990

[25] Wilhelms, J., J. Challinger, N. Alper, S. Ramamoorthy, and A. Vaziri, "Direct Volume Rendering of Curvilinear Volumes," Computer Graphics, Vol. 24, No. 5, November 1990

[26] Williams L., "Pyramidal Parametrics," *Computer Graphics*, Vol. 17, No. 3, July, 1983, pp. 1-11

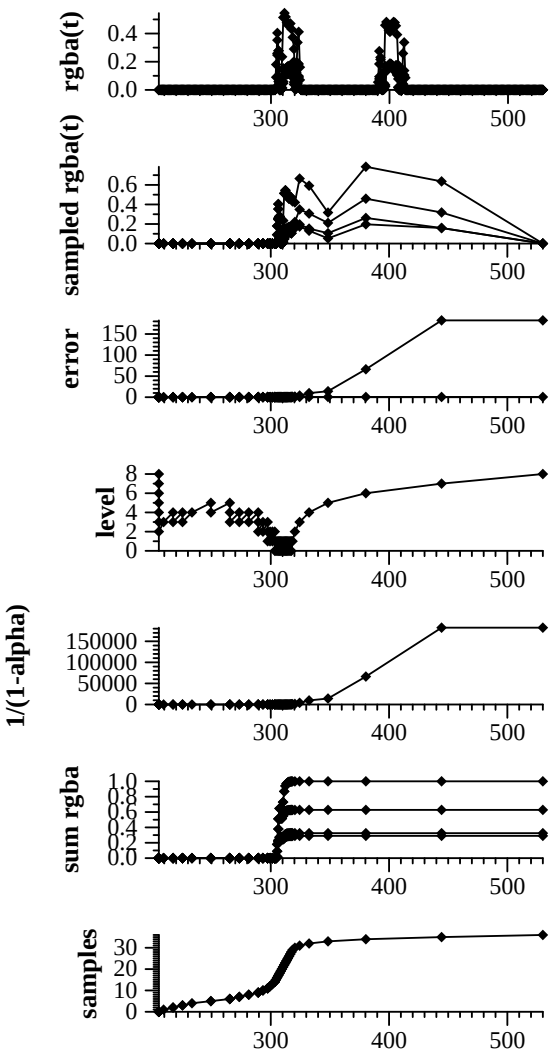

Figure 2: The graph shows the history of a ray tracing through a lobster's claws (Figure 7). The top graph *rbga(t)* plots RGBα values along the ray at full resolution. Next, *sampled rgba(t)* shows the scaled samples as they appear to the *β-acceleration* algorithm. Sample spacing is an indication of speed. *Error* plots the value sampled from the $range_{27}$ pyramid, and the largest "acceptable" value = $k/(1-a)$. The latter is too small to be visible in this graph because of the large amount of accumulated opacity. *Level* plots the level in the pyramid that the ray tracer is sampling in. Vertical lines denote no progress. $1/(1-alpha)$ can be thought of as a scaling factor for k. *Sum rgba* plots the accumulated color and opacity. *Samples* plots the total number of samples. The ray tracer is moving more slowly where the *samples* graph is steep. The ray moves quickly through empty space until it hits a claw: the claw is traced at the lowest resolution until enough opacity is accumulated, and then the ray tracer accelerates exponentially through the rest of the volume.

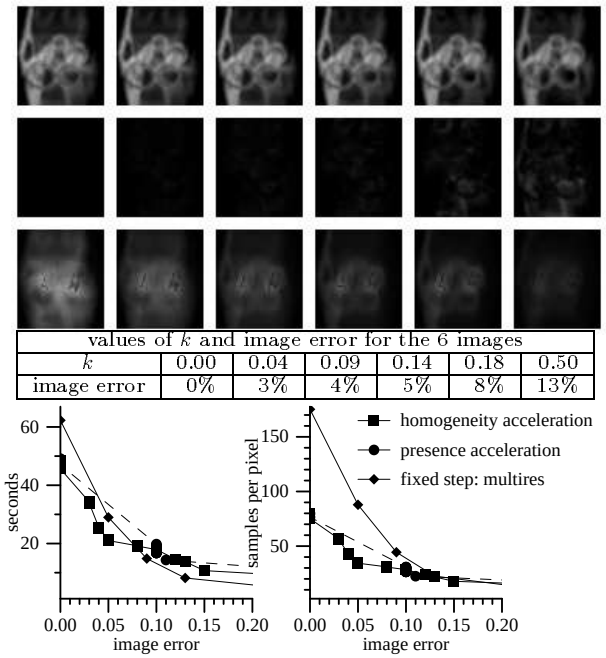

Figure 3: *Homogeneity-acceleration*: Images show performance of *homogeneity-acceleration* for different values of k. Top row displays resulting images, middle row shows absolute value of pixel-wise difference of images from reference image. Ref. image was generated with *Fixed Step Multiresolution* method at full res. Bottom row shows samples per pixel. Plots show performance of *homogeneity-acceleration*, *presence-acceleration*, and *Fixed Step Multiresolution*. Left plot shows seconds per image. Right plot shows samples per pixel. Test images are 56 × 64, dataset is 256 × 256 × 110.

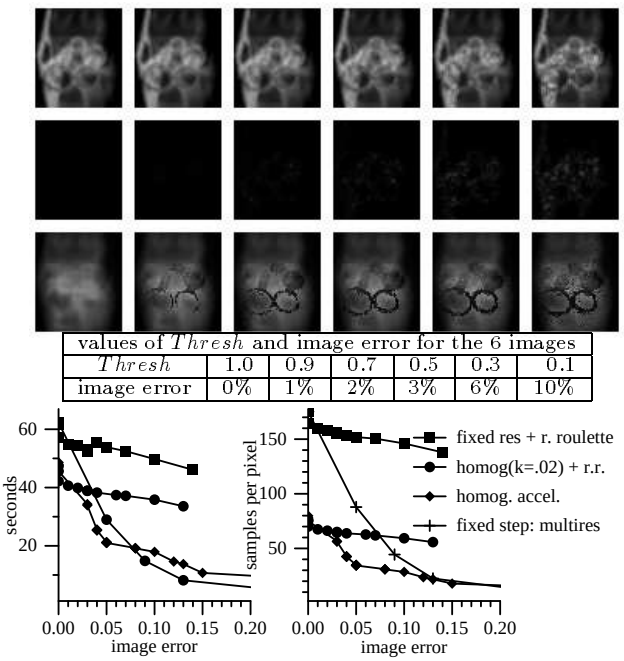

Figure 4: Russian Roulette: images show performance of Russian Roulette combined with *homogeneity-acceleration* for different values of $Thresh$. Top row displays resulting images, middle row shows absolute value of pixel-wise difference of images from reference image. Ref. image was generated with *Fixed Step Multiresolution* at full res. Bottom row shows samples per pixel. Plots show performance of Russian Roulette by itself, Russian Roulette with *homogeneity-acceleration*, plain *homogeneity-acceleration*, and *Fixed Step Multiresolution*. Left plot shows seconds per image. Right plot shows samples per pixel. Test images are 56 × 64, dataset is 256 × 256 × 110.

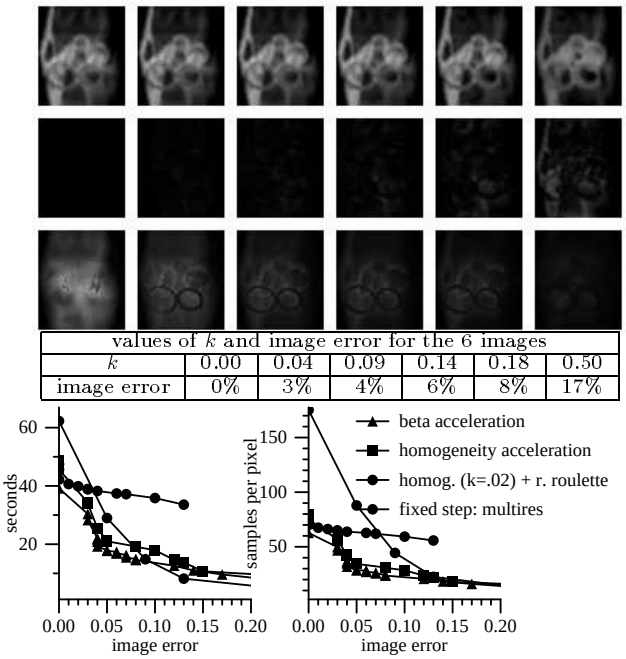

Figure 5: β-Acceleration: images show performance of β-*acceleration*. Top row displays resulting images, middle row shows the absolute value of pixel-wise difference of the images from reference image. Ref. image was generated with *Fixed Step Multiresolution* at full res. Bottom row shows samples per pixel. Plots show performance of β-*acceleration*, *homogeneity-acceleration*, Russian Roulette with *homogeneity-acceleration*, and *Fixed Step Multiresolution*. Left plot shows seconds per image. Right plot shows samples per pixel. Test images are 56 × 64, dataset is 256 × 256 × 110.

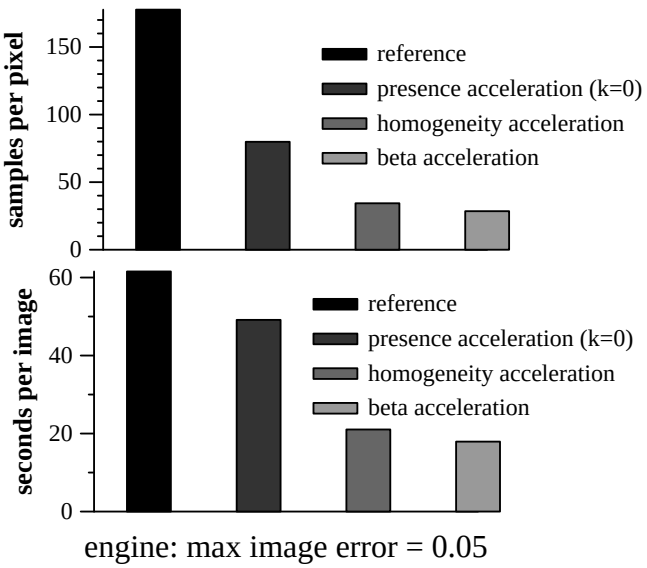

Figure 6: Samples per pixel and seconds per image for (from left) vanilla volume ray tracing, *presence-acceleration* with $k = 0$ (Levoy's algorithm), *homogeneity-acceleration*, and β-*acceleration*. The dataset is the cylinder head shown in Figure 7. The rendering geometry is the same. We took 56 × 64 evenly spaced samples in the image plane.

Figure 7: Lobster: dataset is $320 \times 320 \times 34$. We are looking down the Y axis. Cylinder Head: dataset is $256 \times 256 \times 110$. We are looking diagonally across all of the axes. The left image is the one we used for all of our measurements. The right image is included to aid comprehension