Adaptive Perspective Ray Casting

Kevin Kreeger, Ingmar Bitter, Frank Dachille, Baoquan Chen, and Arie Kaufman[*]

Center for Visual Computing (CVC)
and Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11749-4400, USA

Abstract

We present a method to accurately and efficiently perform perspective volumetric ray casting of uniform regular datasets, called Exponential-Region (ER) Perspective. Unlike previous methods which undersample, oversample, or approximate the data, our method near uniformly samples the data throughout the viewing volume. In addition, it gains algorithmic advantages from a regular sampling pattern and cache-coherent read access, making it an algorithm well suited for implementation on hardware architectures for volume rendering. We qualify the algorithm by its filtering characteristics and demonstrate its effectiveness by contrasting its antialiasing quality and timing with other perspective ray casting methods.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.4.2 [Image Processing And Computer Vision]: Enhancement—Filtering;

Keywords: Volume rendering, Perspective ray casting, Adaptive supersampling, Volume rendering hardware

1 Introduction

Volume rendering is a method for visualizing three dimensional datasets. For realistic visualizations, perspective projections are required unless the viewer's eyepoint is far away from the object being examined. Currently, however, the use of perspective projection either increases the rendering time or decreases the projected image quality. Meanwhile, users' expectations for image quality and speed continue to rise.

Algorithms for volume rendering generally fall into two categories: image-order (e.g., ray casting [7]) and object-order (e.g., splatting [13] or shear-warp [6]). Image-order algorithms are generally thought to produce higher quality images than object-order. However, perspective projections present inherent challenges when performing ray casting.

For parallel projections, the rays that are cast through the volume maintain a constant sampling rate on the underlying volume data. It is straightforward to set this sampling rate to create an output image of the required quality. For perspective projections, however, the rays do not maintain such a continuous and uniform sampling rate. Instead, the rays diverge as they traverse the volume from front to back. This creates an uneven sampling of the underlying volume. Naive ray casting algorithms generally handle ray divergence by one of two methods. The first method is undersampling, in which rays are cast so that the sampling rate at the front of the volume is appropriate for the desired image quality. However, because of the perspective ray divergence, the underlying volume is undersampled. This may result in severe aliasing by creating "holes" in the rear of the volume where regions of voxels remain unsampled. The second method is oversampling, in which rays are cast so that the sampling rate at the rear of the volume is appropriate for the desired image quality. This approach avoids the aliasing of the first method; however, the volume may be radically oversampled in the front. The inefficient oversampling in the front of the dataset dramatically increases the runtime of this method. Of course, the rays can be cast with a sampling rate between undersampling and oversampling. This results in a tradeoff between the image quality of oversampling and the rendering speed of undersampling.

Others have addressed the problem of nonuniform sampling due to perspective ray divergence during volume rendering. Levoy and Whitaker [8] proposed using a 3D-mipmap representation of the underlying volume data to create larger sampling kernels when the rays diverge. However, this method adds a level of blurring before the sampling. Furthermore, the use of non-linear mapping from density-to-color may result in incorrect color being assigned to a region of averaged voxel densities. Swan et al. [12] presented a similar approach for splatting. Novins et al. [9] proposed an adaptive sampling technique in which rays split into four child rays once the neighboring rays diverge past some threshold. Adaptive sampling ensures that the sampling rate remains close to the density of the underlying volume (in fact within the threshold value) without blurring the data and still allowing arbitrary density-to-color mappings. Kreeger et al. [5] proposed two back-to-front merging algorithms similar to Novins et al.'s splitting approach. The first utilized local neighborhood information to decide when to merge two rays, while the second algorithm utilized global information to decide which rays were permitted to merge. However, both algorithms suffered from irregular ray resampling patterns.

Bitter and Kaufman [1] proposed a perspective projection method for the Cube-4 Light volume rendering hardware. Their method resamples the compositing buffer for each volume slice. While this regular resampling operation

[*]{kkreeger,ari}@cs.sunysb.edu

allows the characterization of the filter as a Gaussian, the filter is view angle dependent and spans up to half as many voxels as there are slices in the image for the worst-case of 22.5 degree ray projections. More recently, Brady et al. [2] proposed a two-phase approach which also attempts to sample the volume at a rate near that of the underlying volume. This method divides the viewing frustum into regions based on Euclidean distance from the eyepoint. The distances used as region boundaries are left to be specified by the user. The algorithm then ray casts each region separately with a ray density near the underlying volume resolution. The images created from the regions are subsequently composited by texture mapping hardware. This algorithm is designed for accelerated rendering of large volumes at the expense of purely ideal sampling and filtering. The authors report that when some of the regions are reused to speedup the rendering of subsequent frames, the error from sample approximations becomes difficult to characterize.

To eliminate the random access of volume data during depth-first perspective ray casting, many researchers have proposed to process the volume in a breadth-first slice-order fashion, thereby taking advantage of cache coherency [9, 6, 5]. Kaufman and Bakalash first introduced slice-order processing in 1985 [4] followed by Drebin et al. in 1988 [3] who proposed to have the slices correspond with a scanline in the final image. However, they both required rotating the volume — a process which may take longer than the rendering may take and require twice the memory — so that a slice-by-slice orthographic projection could be performed. Current slice-order volume rendering methods do not require the rotation stage; they cast rays at non-orthographic projections, but still access the data in a slice-by-slice fashion. This powerful method is used by the Cube volume rendering architectures to achieve real-time frame rates with high image quality [11, 1, 10].

2 ER-Perspective Ray Casting

2.1 Overview

We propose an algorithm, *ER-Perspective* (Exponential Regions Perspective), for performing perspective projections of uniform regular datasets by adaptively sampling the underlying volume. Our algorithm provides extremely good antialiasing properties associated with oversampling while giving runtimes on the order of undersampling. Furthermore, it creates at least one sample for every visible voxel in the volume. ER-Perspective gains a runtime advantage over previous work [9, 2, 8] by utilizing slice-order voxel access, while maintaining equal or better image quality. (See Section 4 for details contrasting our algorithm to others.) Figure 1 compares our ER-Perspective algorithm with undersampling for an LGN neuron dataset obtained from a confocal microscope scan. Figures 1(c) and 1(d) show a portion of the image blown up 4 times.

Our ER-Perspective algorithm works by dividing the view frustum into regions based on exponentially increasing distances from the eyepoint. We cast continuous rays from back-to-front (or front-to-back) and merge (or split) the rays once they become too close (or too far) from each other. (We limit the discussion in this paper to the more intuitive case of back-to-front with merging. The differences are pointed out where they are significant.) We use the region boundaries to mark the locations where the rays should be merged. By defining the regions and merging all rays at the boundaries, the algorithm provides a regular pattern of ray merging that

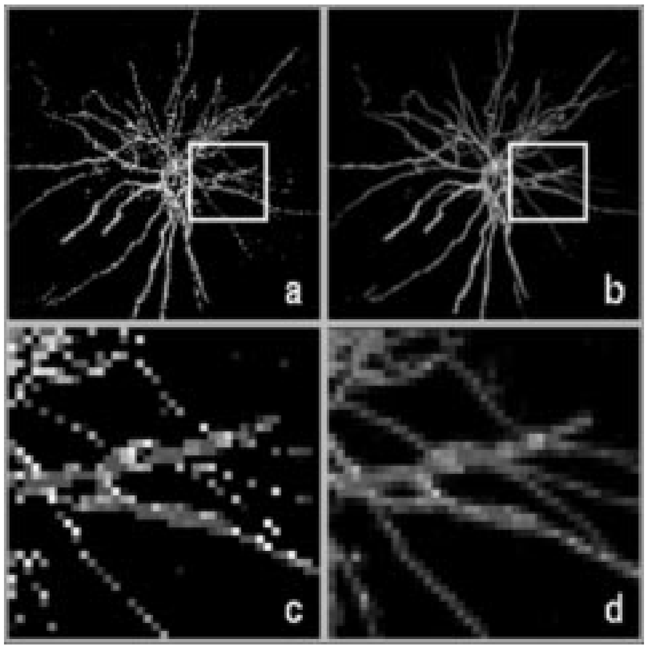

Figure 1: *LGN neuron rendered by (a) Undersampling, (b) ER-Perspective. (c) and (d) are the area in the square blown up 4 times.*

is dependent on the global geometry instead of local neighborhood conditions. Additionally, an odd number of rays are merged such that the resulting ray is an exact continuation of the previous center ray. This is an advantage of our approach over previous methods. It allows us to qualify the algorithm by characterizing the filtering achieved when adaptively sampling the volume (see Section 3.1).

The base sampling rate of the algorithm can be set according to the desired image quality. The base sampling rate is the minimum ray density compared to the underlying volume resolution. (Although the ER-Perspective algorithm supports any sampling rate, for the remainder of this paper we assume that it is 1 ray per voxel.) The algorithm has the advantage of keeping the ray density between 1 and 2 times the base sampling rate. This guarantees that no voxels are missed in the rear of the volume and places an upper bound on the total amount of work performed at two times supersampling.

Since we utilize slice-order processing, we project the volume onto the baseplane of the volume which is most perpendicular to the view direction. The baseplane image is then warped onto the final image plane in the same manner as in shear-warp [6] or Cube-4 [11].

One of the major driving forces of the algorithm was to develop a method which could map to a hardware architecture. Specifically, we strived to create an algorithm which only required nearest neighbor communication between processing elements. While processing a row of voxels on a one-dimensional array of processing elements, our algorithm only requires processing elements to communicate with their immediate left and right neighbors.

2.2 Exponential Region Selection

Our algorithm uses slice-order processing along one of the three major axes. Consequently, we define the regions in our algorithm as slabs of slices along the major projection

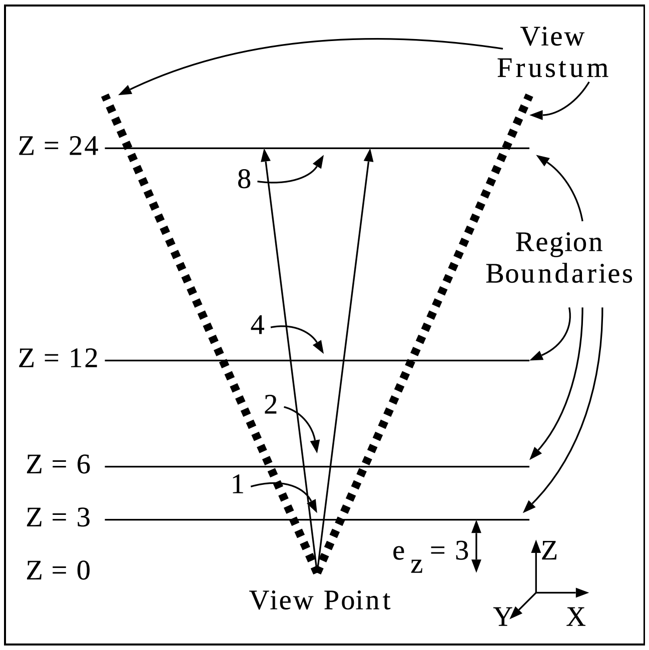

Figure 2: *Exponential region boundaries in voxel units. The two perspective rays have the desired property that they are twice as far apart at the rear boundary of each region as they are apart at the front boundary of each region.*

axis. (For the remainder of the paper we assume that the volume is being projected along slices perpendicular to the Z-axis.) Specifically, we take the distance along the Z-axis from the viewpoint to the front of the volume and create the first region to consist of as many Z-slices as this distance. Each successive region after the first one is twice as deep as the one before it.

To illustrate, if the viewpoint is 3 voxel units in front of the volume, then the first region is 3 voxel units thick, the next is 6 voxel units thick, etc. In general, the i-th region is $e_z \cdot 2^i$ slices thick, where e_z is the distance from the viewpoint to the front of the volume (see Figure 2). Forcing the regions to be thus defined produces the desired effect that any two perspective rays shot through any of the regions are twice as far apart at the rear boundary as they are at the front boundary. This is shown in Figure 2 as the distance between the two rays grows from 1 to 2 across the first region, then to 4, and finally to 8 at the rear of the last region. Additionally, since the region boundaries are dependent on the global geometry, the efficiency of the ray casting algorithm is maximized by providing a mechanism for keeping the ray density between 1 and 2 times the underlying volume resolution in each dimension. It also creates a regular topology so that the filtering of the data can be controlled as perspective rays are cast.

2.3 Ray Density Resampling

Having regions with boundaries at exponential distances produces ray density twice as high at the front as at the back of the region. Therefore, we must provide a mechanism to adjust the ray density when crossing a region boundary. Since each ray starts on a voxel coordinate at the rear of a region, at the front of the region every second ray in each dimension will, once again, coincide directly with a voxel coordinate. The remaining rays intersect the region boundary halfway between two voxel positions. To downsample the ray density with this deterministic ray pattern, we use a 2D Bartlett filter (also known as tent or triangle filter) with an extent of ± 1 voxel unit in each dimension [14]. Because the ray density at the front of each region is twice the voxel density, this 3×3 voxel neighborhood is intersected by 5×5 rays. Since the edges have a weight of zero (see Figure 3), only the 3×3 neighboring rays are used for applying the filter to downsample the ray density. This effectively merges neighboring rays. The Bartlett filter was chosen over a simple box filter for the added quality it produces in the final image. (See Section 3.1 for our analysis of the cascading of local Bartlett filters.) For front-to-back processing, rays are split instead of merged. Here a bilinear interpolation of the rays is performed to generate the new rays which begin between other rays. Note that the Bartlett filter of size ± 1 is the inverse of the bilinear interpolation operation.

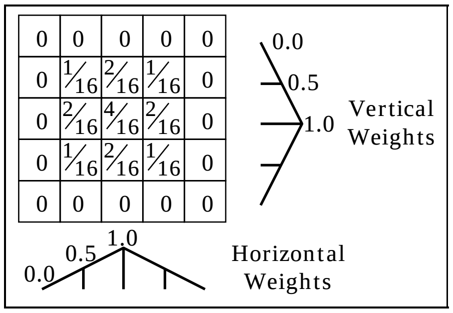

Figure 3: *2D filter of size ± 2 samples. The filter is a linear ramp in each dimension. To generate a 2D filter, the weights are multiplied and then normalized.*

2.4 Adaptive Ray Casting

Notice in Figure 4 that the volume does not need to end on a region boundary. However, since we want the rays to be on exact voxel coordinates at all of the region boundaries, we begin the rays on the grid coordinates at the rear of the last enclosing region. Therefore, the voxel coordinates and the ray sample locations may not be congruent at the rear of the volume. This not only provides the mentioned boundary conditions, but aids with temporal antialiasing when the eyepoint is moved in smaller than voxel unit distances because the rays will continue to originate from the same positions relative to the voxels.

Algorithm 1 performs the ER-Perspective back-to-front projection of a volume. First, the exponential boundaries are created for the regions given the eye position in voxel units. We establish enough regions to completely encompass the volume. To perform the rendering, we loop through each region from the back to the front, computing normal ray casting, but in a slice-order fashion, and store the partially computed rays in a compositing buffer. Between regions we perform the ray density resampling of the compositing buffer described in Section 2.3. The baseplane image is warped onto the final image plane for display.

Figure 4 shows a 2D example of how the rays travel through a 7^3 volume when the viewpoint is 3 voxel units in front of the volume. Notice that the sampling rate is always between 7 and 14 per slice, and that it increases as the rays travel through the regions from back to front. The number of ray density resampling stages for an N^3 volume is limited by $log_2 N$, since that is the maximum number of regions in an N^3 volume. The last resampling step shown

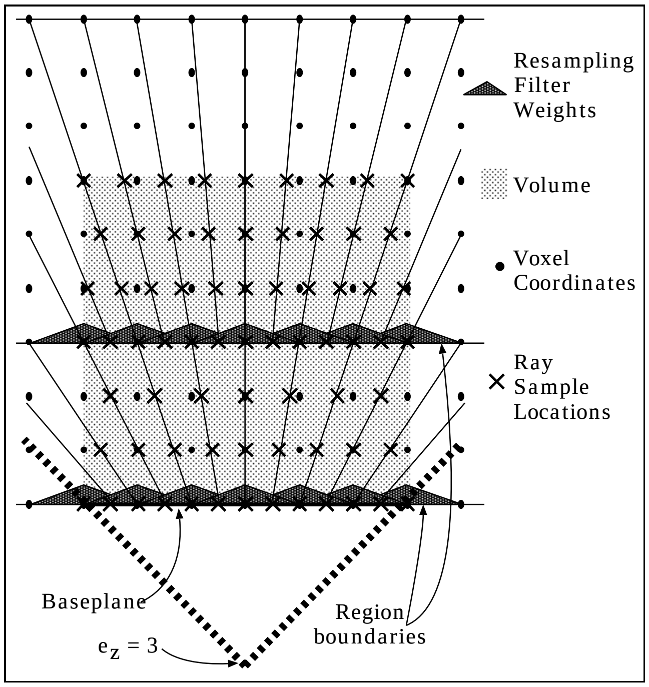

Figure 4: *A 7^3 volume, where the viewpoint is 3 voxel units in front of the volume. The rear of the volume does not coincide with a region boundary, but the rays are still positioned on the voxel coordinates at the rear of the region boundary. In this example the view frustum is set up so that the final image coincides with the baseplane*

on the baseplane is performed when the image warp takes place.

3 Evaluation of the Algorithm

3.1 Qualifying the Filter

With previous adaptive ray density perspective methods, it was difficult to determine the filtering function achieved when rays were merged using irregular patterns. Since we use regular boundaries for the filtering operations and exact ray placement within the boundaries, it is possible to compute the effective filter achieved by the cascading of local Bartlett filters. This is one of the major advantages of our ER-Perspective algorithm. Additionally, we show that the boundaries and filter we have chosen overcome the poor image quality usually associated with successive filtering of discrete data.

Consider the case of a perspective projection of a volume 7 slices deep with the eyepoint 2 voxel units in front of the volume, as depicted in Figure 5. Using our ER-Perspective approach, the rays that are cast through the region are one voxel unit apart at the rear of the region. However, the rays reach a region boundary and are filtered using local Bartlett filters. The Bartlett filter (simplified to 1-dimension) contains the following weights for a kernel of size $2n+1$ normalized so that the output has the same scalar range as the input:

$$0, \frac{1}{n^2}, \frac{2}{n^2}, \cdots, \frac{n-1}{n^2}, \frac{n}{n^2}, \frac{n-1}{n^2}, \cdots, \frac{2}{n^2}, \frac{1}{n^2}, 0 \tag{1}$$

The ER-Perspective algorithm always resamples the rays to

```
Compute Z-position of Eye in Voxel Units
Compute Exponential Region Boundaries
for REGION = MaxRegion to 0
  for SLICE = MaxSlice[REGION] to MinSlice[REGION]
    Bilinear Interpolate Samples for this slice
    Shade and Classify Samples
    Composite Samples onto Rays in Buffer
  end for
  if not frontmost REGION
    Downsample Rays in Compositing Buffer with
      Bartlett Filter
  end if
end for
Warp Baseplane to Final Image plane
```

Algorithm 1: *Back-to-Front ER-Perspective Ray Casting (assuming Z-major axis projection)*

half of the original density. Using a filter of size ± 2 rays (n=2) creates a filter kernel of 5x5, or just the following 5 weights for one dimension:

$$0, \frac{1}{4}, \frac{2}{4}, \frac{1}{4}, 0 \tag{2}$$

Now, consider the contribution of samples a, b, c, d and e to the partially composited ray which changes from region 2 to region 1 at location o,

$$o = \frac{1}{4}b + \frac{2}{4}c + \frac{1}{4}d \tag{3}$$

likewise the partial rays at p and q are computed

$$p = \frac{1}{4}d + \frac{2}{4}e + \frac{1}{4}f \tag{4}$$

$$q = \frac{1}{4}f + \frac{2}{4}g + \frac{1}{4}h \tag{5}$$

(We omit the formulas for partial rays for n and r since they have a 0 weight in the final filter for pixel A.) Continuing the ER-Perspective algorithm, the resampled partial rays n, o, p, q and r are cast through region 1 where they are again filtered by a local Bartlett filter. Now, the normalized contribution of n, o, p, q and r to pixel A is:

$$A = \frac{1}{4}o + \frac{2}{4}p + \frac{1}{4}q \tag{6}$$

Substituting in the values for o, p and q gives us:

$$A = \frac{1}{16}b + \frac{2}{16}c + \frac{3}{16}d + \frac{4}{16}e + \frac{3}{16}f + \frac{2}{16}g + \frac{1}{16}h \tag{7}$$

Notice that this formula contains the same weights as a Bartlett filter with kernel size of nine values (n=4). This can be repeated for pixel B with the same filter weights. For front-to-back processing a similar analysis can be used to demonstrate the performance of the algorithm and the result of successive applications of the bilinear interpolation.

We can also show that each sample of a slice contributes the same amount to the final image as any other sample in the same region (assuming all other operations on samples, such as color mapping and compositing, are equal). For example, the value sample e contributes to pixel A with an effective weight of $\frac{1}{4}$ after the cascading of the local Bartlett

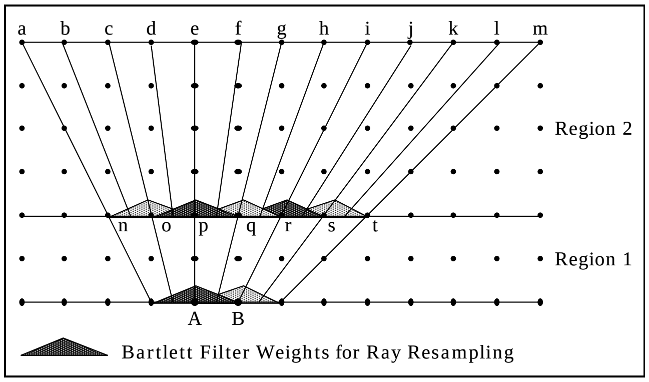
a b c d e f g h i j k l m
Region 2
n o p q r s t
Region 1
A B
Bartlett Filter Weights for Ray Resampling

Figure 5: *ER-Perspective ray casting across two regions.*

filters. Likewise, sample i contributes to pixel B with an effective weight of $\frac{1}{4}$. Sample f contributes to pixel A with a weight of $\frac{3}{16}$ and to pixel B with a weight of $\frac{1}{16}$ for a total of $\frac{1}{4}$. This can be repeated for samples g and h. The samples to the left of e and the right of i contribute partly to pixels left of A and right of B, respectively, so that the sum of their contributions to the final image is also $\frac{1}{4}$. In fact, every sample that is in this region has the same weight. The weight is $\frac{1}{4}$ because this region is the second region in the volume. For the first region in the volume, every sample has a weight of $\frac{1}{2}$. This is qualifiable by realizing that there are 2 rays per final image pixel in this region. There are 4 rays per final image pixel in the second region, etc. Consequently, the weight which determines the contribution of each sample towards the final image is the ratio $\frac{\text{image pixels}}{\text{samples in this slice}}$.

3.2 Image Quality

Two-dimensional texture mapping has long utilized challenging images for measuring aliasing and comparing methods. The images that are often used contain high frequency components such as checkerboard patterns of black and white squares. We propose to use a similar idea to create test volumes for measuring the effectiveness of algorithms to accurately render antialiased volumes. Figure 6 (Figure 12 in the Color Section) shows images of our volume dataset for measuring antialiasing performance of perspective algorithms. The dataset consists of planes of small cubes in a checkerboard pattern. The volume consists of a floor of blue cubes and left and rear walls of white cubes. The cubes are 5^3 voxels and the volume is 256^3.

The images in row (a) of Figure 6 were rendered by an undersampling algorithm where the ray density is set to be equal to the underlying volume at the front of the volume. In these images, ray divergence causes two types of aliasing. The first is Moire patterns due to the non-uniform sampling, which occurs on the walls which are perpendicular to the view plane. The second is the total elimination of small features towards the rear of volumes due to perspective ray divergence. This aliasing occurs on the rear wall which is parallel to the view plane. Row (b) of this figure was created using our ER-Perspective algorithm. Note that there are no noticeable aliasing effects. Figure 6 row (c) was created using oversampling, where the ray density is equal to the underlying volume at the rear of the volume. Because of ray divergence, the volume is oversampled at the front. The images were rendered so that the image plane coincided with the front of the volume. The left column is rendered with the

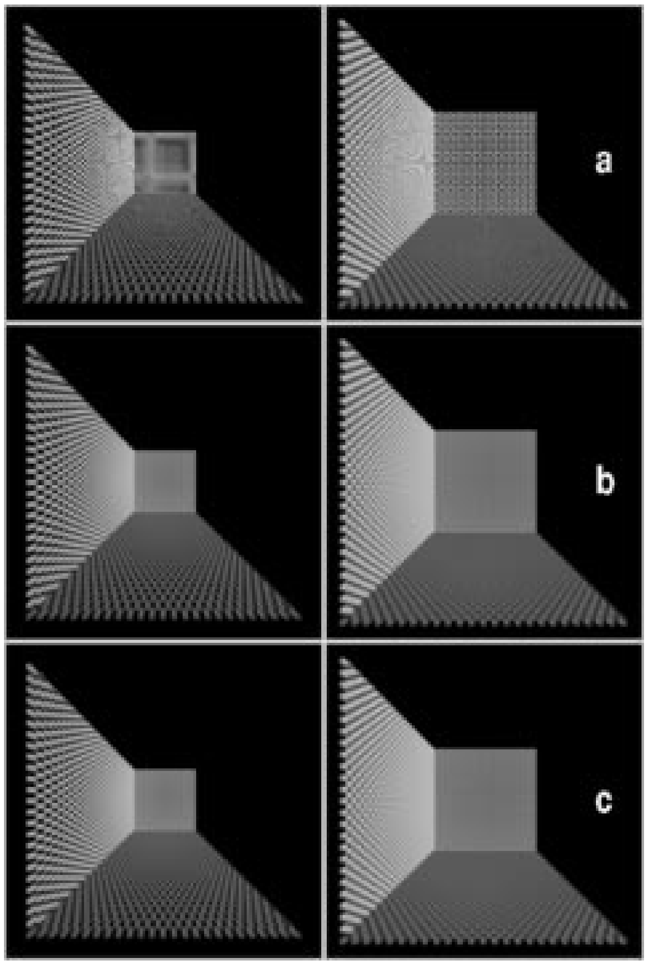

Figure 6: 256^3 *test volumes with* 5^3 *subcubes. Row (a) Undersampling, (b) ER-Perspective, (c) Oversampling. The left column is rendered with the eye at (128,128,-64) and a* 127 deg *field-of-view, while the right column is with an eye at (128,128,-128) and* 90 deg *field-of-view.*

eye at (128,128,-64) and a 127 deg field-of-view. The right column from (128,128,-128) and 90 deg.

Figure 7 (Figure 12 in the Color Section) shows the undersampling, ER-Perspective and oversampling methods for the test volume with a different subcube granularity and an LGN neuron dataset. Our new exponential perspective method does not suffer from any of the aliasing artifacts for these datasets either. Notice that the quality of our projections visually matches the quality of the much slower oversampling method. In the LGN pictures, notice the ganglions which extend towards the rear of the neuron. With the undersampling method, there is noticeable disappearance of these small features which could greatly affect the interpretation of the dataset.

Note that there is no noticeable visual difference between the images created with our ER-Perspective algorithm and the oversampling method. Figure 8 is the difference image between the ER-Perspective method and the oversampling method for the test volume in Figure 7. The difference image is scaled up to make the values displayable. The actual range of the differences found is 5 out of 256 grayscales (the R, G and B difference images with 256 values each was converted to grayscale).

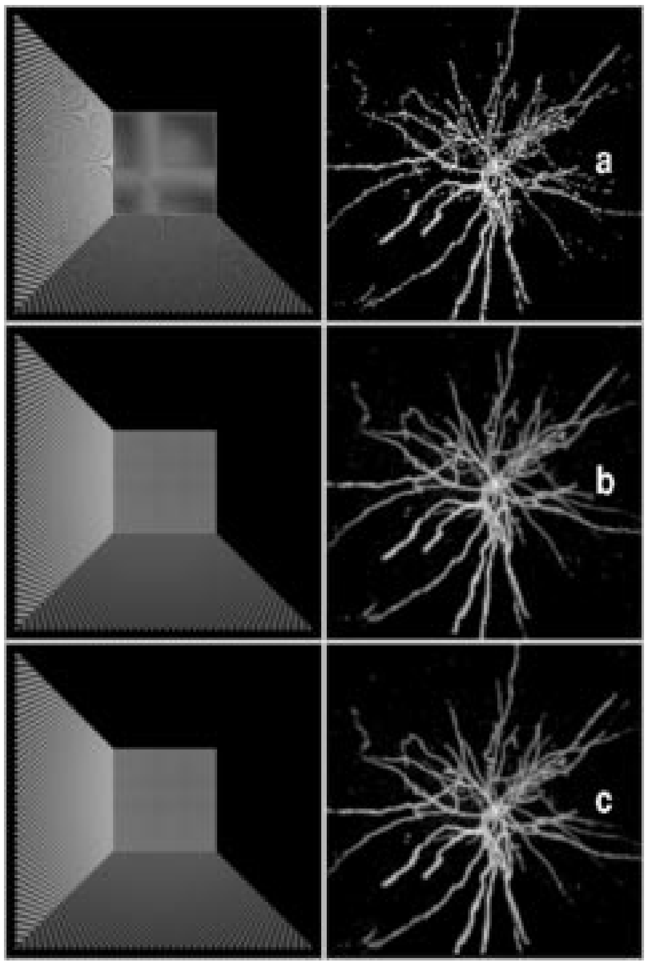

Figure 7: *The left column is a* 256^3 *test volumes with* 3^3 *sub-cubes with the eye at (128,128,-128) and* 90 deg *field-of-view. The right column is the LGN dataset. Row (a) Undersampling, (b) ER-Perspective, (c) Oversampling.*

The region of the oversampling image which represents the front half of the volume in Figure 7 differs slightly from the ER-perspective image in the same figure because of the higher sampling rate performed by the oversampling method. This cannot be avoided when sampling at different rates. However, the regions of the same images which represent the rear half of the volume are identical. This gives empirical proof that the cascading of Bartlett filters is equivalent to one wide Bartlett filter as shown in Section 3.1.

3.3 Performance

Since we are performing a slice-order algorithm, it is simple to analyze the total amount of computation by calculating the amount of work performed on each slice. Assuming that the work done on each sample is the same, the count of the number of samples processed can be used as a comparison of the workloads. For example, in the oversampling method, the number of samples on the rear slice of a volume which ends exactly on a region boundary is N^2. On the front slice, the sample count depends on the geometry of the eyepoint. In particular, using similar triangles and e_z as the distance of the eyepoint from the front of the volume, the number of

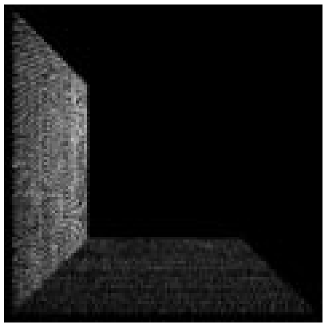

Figure 8: *Difference between Oversampling and ER-Perspective from Figure 7 scaled up for display-ability. The actual range of the differences is 5/256 (the RGB image was converted to grayscale).*

samples taken is

$$\left(\frac{N^2 + N \cdot e_z}{e_z}\right)^2 \tag{8}$$

This can be generalized for any slice s through the volume to

$$\left(\frac{N^2 + N \cdot e_z}{e_z + s}\right)^2 \tag{9}$$

Thus, the total count of samples processed by the oversampling method is

$$\sum_{s=0}^{N} \left(\frac{N^2 + N \cdot e_z}{e_z + s}\right)^2 \tag{10}$$

Similarly, the undersampling method can be shown to perform the following amount of work

$$\sum_{s=0}^{N} \left(\frac{N \cdot e_z}{e_z + s}\right)^2 \tag{11}$$

For our ER-Perspective algorithm the analysis is more complicated. Depending on the viewing geometry, we create $\log\left(\frac{N+e_z}{e_z}\right) - 1$ regions. We have shown in Section 2.2 that each of these regions has $e_z \cdot 2^i$ slices. Again using similar triangles, our ER-Perspective algorithm processes the following number of samples

$$\sum_{reg=0}^{\log\left(\frac{N+e_z}{e_z}\right)-1} \sum_{s=0}^{e_z * 2^{reg}} \left(\frac{N * (e_z * 2^{reg} - e_z + s)}{e_z * 2^{reg} - e_z}\right)^2 \tag{12}$$

This complicated formula has an upper bound of

$$\sum_{s=0}^{N} (2N)^2 \tag{13}$$

and a lower bound of

$$\sum_{s=0}^{N} (N)^2 \tag{14}$$

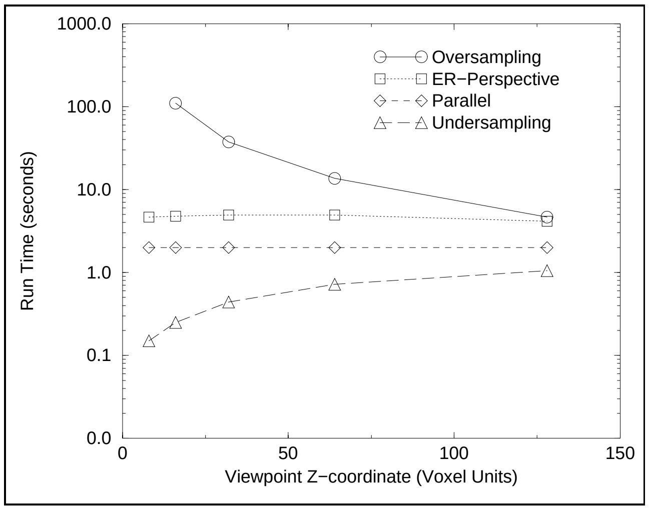

Figure 9: *Runtimes for each of the algorithms for a* 125^3 *LGN neuron volume as the distance from the viewpoint to the front of the volume changes. The algorithms ran with a high-quality full Phong equation look-up-table shading of every sample point.*

Examining Equation 10 we can see that the oversampling method could perform $O(N^4)$ work on the front slice when the eyepoint is very close to the volume. Figure 9 presents measured runtimes for the various rendering algorithms at several viewpoints (all implemented with a high-quality Phong shading look-up-table). The oversampling times grow rapidly as the eyepoint is moved closer to the front of the volume.

If we examine Equation 11, we see that as the eyepoint approaches the front of the volume, the numerator approaches zero. The amount of work performed on the rear slice also approaches zero. We can, once again, see in Figure 9 that the runtimes of the undersampling method decrease as the eyepoint becomes closer to the volume.

Equations 13 and 14 show that regardless of the viewpoint geometry, the amount of work performed by our ER-Perspective algorithm is bounded by $\Omega(N^2)$ and $O(4N^2)$ per slice. This provides the following two advantages:

- an upper bound on the runtime of the algorithm that is linear with the number of voxels and is independent of the view position
- a lower bound on the image quality achieved that is also independent of the view position

Thus, the user can set the base sampling rate (see Section 2) for the desired image quality and be sure that the sampling rate is sufficient throughout the volume for that quality.

In contrast, the oversampling method provides a lower bound on the image quality yet the runtime of the algorithm may become much greater than that of the ER-Perspective. The undersampling method provides an upper bound on the runtime of rendering, but the image quality may become much worse than the ER-Perspective.

The algorithm is also parallelizable. We created a simple parallel version of our code to run on 8 PEs of an SGI Challenge using threads and SMP features. Each PE classified, shaded and composited $\frac{1}{8}$ of each slice. With no load balancing, the algorithm rendered images of the 125^3 LGN neuron in 1.75 seconds. A proper parallel version with good load balancing would perform even better.

4 Comparison To Other Algorithms

In this section we compare and contrast our approach to other algorithms. Levoy and Whitaker's 3D-mipmap approach [8] handled perspective ray divergence by large sampling kernels. To improve the runtime of their algorithm, they precomputed a 3D mipmap of the volume. Thus they have sample kernels of various sizes available during rendering. Our algorithm, on the other hand, effectively classifies each sample and then computes the larger kernel on the fly by merging the rays as they project towards the front of the volume. Levoy and Whitaker's algorithm requires a pre-classification and shading step even before the 3D mipmap generation.

Brady et al.'s two phase approach [2] could be considered a more general case of our algorithm. While we define very specific region boundaries, Brady et al. allow the user to set them. If Brady et al.'s algorithm were utilized with exponentially increasing region sizes, the main difference between our algorithm and theirs is that we utilize a slice-order approach. Therefore, our region boundaries coincide with volume slices while Brady et al.'s boundaries are arcs of equal distance from the eyepoint. We note that Brady et al. utilize shading approximations on their algorithm. For Comparison purposes, we adapted our algorithm to implement the same shading quality as they did. Brady et al. reported 1.35 seconds for a complete raycast from scratch of a 80x80x127 volume on one 300MHz Pentium II. For the same sized volume our algorithm, implemented with the same shading algorithm as Brady et al.'s, rendered in 0.91 seconds on one 195MHz R10000. Even though Brady et al. utilized a faster machine, our algorithm was faster because the slice-order volume access allows for cache-coherency. Additionally, although Brady et al. takes advantage of a 3D graphics coprocessor to perform the inter-region filtering, we have fewer regions to filter since we utilize exponentially growing regions. In essence, our algorithm filters the rays only when the ray density dictates it (and we have found this to be at exponentially increasing distances), while Brady et al. creates regions that are each 16 samples in length. We have also found that cache performance is very important to volume rendering runtimes. Our machine has a 2MB L2 cache.

The algorithm by Novins et al. [9] utilizes a box filter to merge the split child rays. Wolberg [14] has shown that the box filter contains more prominent side lobes in the stopband of the frequency domain than the Bartlett filter and therefore contributes to more aliasing. Since Novins et al. designed their algorithm in 1990, we re-implemented it on our hardware to compare the runtimes. The results show that our algorithm is twice as fast, attributed to the fact that we utilize slice-order volume samples. Additionally, we observed that the box filter, while removing most of the aliasing artifacts of the undersampling method, left more aliasing than the ER perspective algorithm (see Figure 10 and Figure 11 in the Color Section). This is more noticeable in animations. We notice similarities to our algorithm despite the slice-order sampling and the filter used for splitting/merging. Specifically, Novins et al. split whenever the ray density becomes less than 1 ray per unit voxel. Our analysis shows that this occurs at exponentially increasing Euclidean distances from the eyepoint, although with an arc pattern similar to Brady et al.'s rather than slice-order like ours.

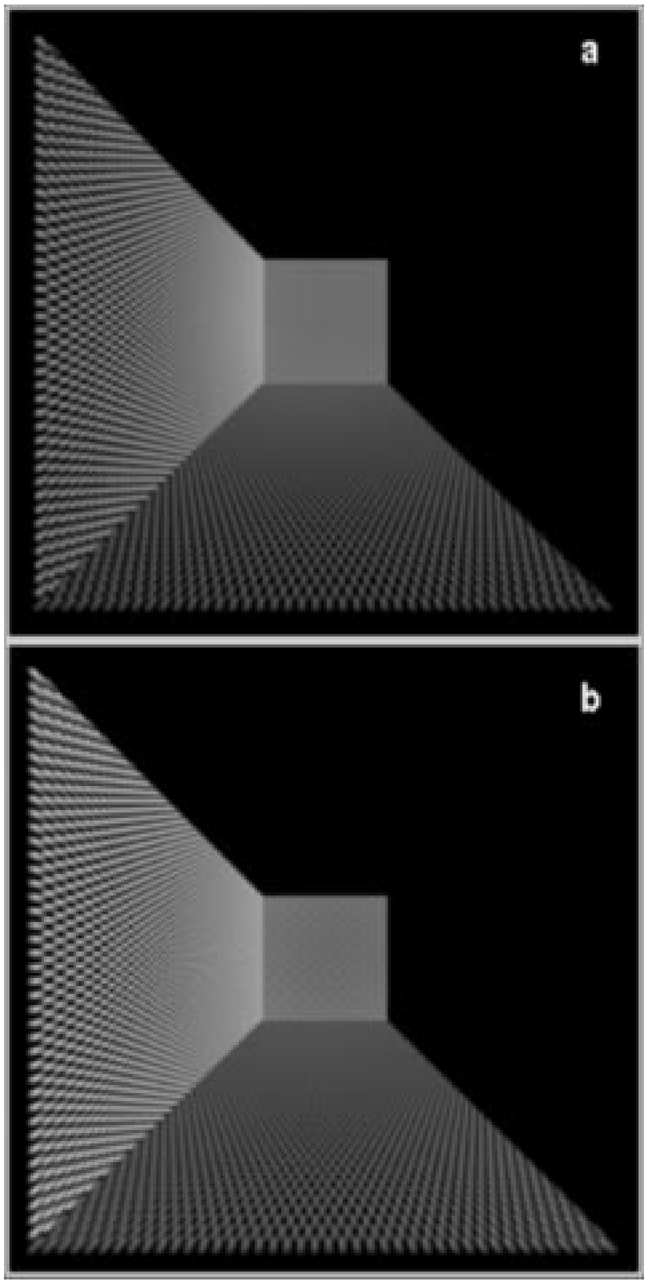

Figure 10: *Rendering the test volume with (a) ER-Perspective with Bartlett filter and (b) Novins et al. algorithm with a box filter. There is slight, but noticeable, reduced aliasing for the Bartlett filter.*

5 Concluding Remarks

We have presented a new algorithm which does not suffer from the traditional pitfalls when performing perspective projections on uniform regular grids. It runs faster than oversampling methods and produces better quality images than undersampling methods. We have shown that our algorithm is a special case of Brady et al.'s more general approach and that slice-order voxel access allows our algorithm to run faster. We have shown that our Bartlett filter for ray merging provides an image quality improvement over the box filter utilized by Novins et al. We have qualified our algorithm by characterizing the effective filtering on the input data. We also have utilized challenging datasets and shown the algorithms antialiasing effectiveness. Our algorithm is attractive because it is very well suited for implementation on hardware architectures to produce real-time perspective volume rendering, such as on the Cube family of architectures. We plan to create the exact mapping of the algorithm on the Cube hardware pipelines and study the VLSI requirements for memory and communication.

6 Acknowledgements

This work was supported by the National Science Foundation under grant MIP9527694, Office of Naval Research under grant N000149710402, Mitsubishi Electric Research Lab, Japan Radio Co., Hewlett-Packard, and Intel Corp. The LGN dataset is courtesy of Howard Hughes Medical Institute, Stony Brook, NY.

References

[1] I. Bitter and A. Kaufman. A Ray-Slice-Sweep Volume Rendering Engine. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 121–130, Los Angeles, CA, Aug. 1997. ACM.

[2] M. Brady, K. Jung, H. Nguyen, and T. Nguyen. Two-Phase Perspective Ray Casting for Interactive Volume Navigation. In *Visualization '97*, pages 183–189, Pheonix, AZ, Oct. 1997. IEEE.

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. In *Computer Graphics, SIGGRAPH 88*, pages 65–74, Atlanta, GA , Aug. 1988. ACM.

[4] A. Kaufman and R. Bakalash. A 3-D Cellular Frame Buffer. In *Eurographics '85*, pages 215–220, Nice, France, Sept. 1985. Eurographics.

[5] K. Kreeger, F. Dachille, and A. Kaufman. A slice-order ray-merging method for volume rendering. Technical Report CVC.97.04.08, Center for Visual Computing, SUNY Stony Brook, NY 11794-4400, Apr. 1997.

[6] P. Lacroute and M. Levoy. Fast Volume Rendering using a Shear-warp Factorization ot the Viewing Transform. In *Computer Graphics, SIGGRAPH 94*, pages 451–457, Orlando, FL, July 1994. ACM.

[7] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(5):29–37, May 1988.

[8] M. Levoy and R. Whitaker. Gaze-directed volume rendering. *Computer Graphics*, 24(2):217–223, Mar. 1990.

[9] K. L. Novins, F. X. Sillion, and D. P. Greenberg. An efficient method for volume rendering using perspective projection. *Computer Graphics*, 24(5):95–100, Nov. 1990.

[10] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 131–138, Los Angeles, CA, Aug. 1997. ACM.

[11] H. Pfister and A. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Visualization. In *Symposium on Volume Visualization*, pages 47–54, San Francisco, CA, Oct. 1996. ACM.

[12] J. E. Swan, K. Mueller, T. Moller, N. Shareef, R. Crawfis, and R. Yagel. An Anti-Aliasing Technique for Splatting. In *Visualization '97*, pages 197–204, Pheonix, AZ, Oct. 1997. IEEE.

[13] L. Westover. Footprint Evaluation for Volume Rendering. In *Computer Graphics, SIGGRAPH 90*, pages 367–376, Dallas, TX, July 1990. ACM.

[14] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.