Opacity-Weighted Color Interpolation for Volume Sampling

Craig M. Wittenbrink, Tom Malzbender, Michael E. Goss
Computer Systems Laboratory
HPL-97-31 (R.2)
July, 1998

E-mail: [craig_wittenbrink;malzbend;gross]@hpl.hp.com

volume rendering,
compositing,
ray tracing

Volume rendering creates images from sampled volumetric data. The compute intensive nature of volume rendering has driven research in algorithm optimization. An important speed optimization is the use of preclassification and preshading. We demonstrate an artifact that results when interpolating from preclassified or preshaded colors and opacity values separately. This method is flawed, leading to visible artifacts. We present an improved technique, opacity-weighted color interpolation, evaluate the RMS error improvement, hardware and algorithm efficiency, and demonstrated improvements. We show analytically that opacity-weighted color interpolation exactly reproduces material based interpolation results for certain volume classifiers, with the efficiencies of preclassification. Our proposed technique may also have broad impact on opacity-texture-mapped polygon rendering.

Internal Accession Date Only

To be published in *IEEE, 1998 Symposium on Volume Visualization*, October 19-20, 1998, Research Triangle Park, North Carolina,

Opacity-Weighted Color Interpolation For Volume Sampling

Craig M. Wittenbrink, Thomas Malzbender, and Michael E. Goss*

Hewlett-Packard Laboratories, Palo Alto

Abstract

Volume rendering creates images from sampled volumetric data. The compute intensive nature of volume rendering has driven research in algorithm optimization. An important speed optimization is the use of preclassification and preshading. We demonstrate an artifact that results when interpolating from preclassified or preshaded colors and opacity values separately. This method is flawed, leading to visible artifacts. We present an improved technique, opacity-weighted color interpolation, evaluate the RMS error improvement, hardware and algorithm efficiency, and demonstrated improvements. We show analytically that opacity-weighted color interpolation exactly reproduces material based interpolation results for certain volume classifiers, with the efficiencies of preclassification. Our proposed technique may also have broad impact on opacity-texture-mapped polygon rendering.

Keywords: volume rendering, compositing, ray tracing.

1 INTRODUCTION

Volume rendering as introduced by Drebin et al. [DCH88] demonstrated classification and shading. Other research has focused on approximation algorithms for speed and parallelization approaches for throughput. There remain subtleties in the proper development of volume rendering systems, both software and hardware. In this paper we examine the approach of rendering from preclassified or preshaded colors, because a potential for artifacts exists. Drebin et al. used opacity-weighted colors, but Levoy [Lev88,Lev90] proposed a simpler approach that directly interpolates colors and opacities. Using non-opacity-weighted colors may create errors.

The artifacts are subtle and can be confusing. We present a technique, ***opacity-weighted color interpolation***, that avoids the artifacts and is an improvement upon Drebin et al.'s approach.

Levoy [Lev88,Lev90] proposed a volume pipeline that classifies and shades voxels before interpolation. This approach can lead to two computational advantages compared to classifying sample points interpolated from voxel data. First, classification can be treated as a preprocess and need not be repeated per view. Second, the number of voxels in a dataset can be significantly less than the number of sample points when computing a view, resulting in lower classification cost. Figure 1 shows a volume rendering pipeline performing separate interpolation of colors and opacities.

Acquired scalar values, $f(x)$, are classified to opacities, $\alpha(x)$, and shaded to compute voxel colors, $C(x)$, at sample points within the volume. The colors and opacities are then

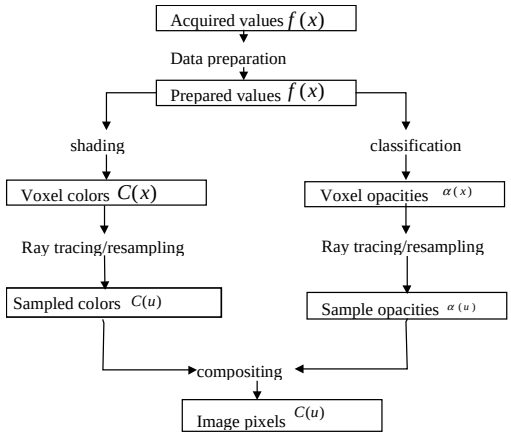

Figure 1. Pipeline analogous to Lev88 "Figure 1. Overview of volume rendering pipeline".

resampled along ray points projecting into the volume, Figure 2A, and composited along the ray to compute a final ray color. We describe how to correctly perform the interpolation, and illustrate why the published technique produces artifacts. There are also implications for hardware which are not obvious, that we analyze and clarify. We first review the optical model and the existing technique, illustrate the artifact, then present our solution. We also evaluate the RMS error improvement, hardware and algorithm efficiency, and demonstrated improvements for several rendering scenarios. In order to understand Figure 1, we need to discuss the optical model and our notation for colors and opacities.

1.1 Volume Rendering Optical Model

Volume rendering algorithms are based on the physics of light interaction with modeled particles in a volume. The beginnings of a particle model for graphics rendering were developed by Blinn [Bli82] and extended by Kajiya et al. [KH84] to nonhomogeneous densities of particles. Blinn adapted the radiative transfer theory particle model for realistic image synthesis [Bli82] from Esposito [Esp79] Chandresakar [Cha60] and others. Blinn's analytical solutions solved the lighting of layers of homogeneous particles, to render Saturn's rings. The common algebraic technique for combining ray samples is

*1501 Page Mill Road, MS 3U4, Palo Alto, CA 94304, {craig_wittenbrink,malzbend,goss}@hpl.hp.com.

compositing, derived separately by Blinn [Bli82] and Porter et al. [PD84]. Kajiya provided a generalization to nonhomogeneous volumetric media, and developed single and multiscattering solutions [KH84]. Modern direct volume rendering algorithms are largely based on Levoy [Lev88], Drebin et al. [DCH88], Sabella [Sab88], and Upson et al. [UK88]. (Note: the work in [DCH88] was demonstrated a couple of years prior to publication, but not released due to its proprietary nature to Pixar). Max has published a survey of direct volume solution approaches [Max95]. Non-particle model volumetric approaches were widely developed before 1988, including [Mea82,Mea85,Kau91]. Other related work in particle and radiative transfer theory applied to volume visualization includes [Max86][KK89].

As direct solution of the posed integro-differential equations for the radiative transfer of light through a nonhomogeneous model is not in general analytically solvable [KH84][Max95], and numerically very expensive to solve, researchers have developed numerous approximations for computational tractability. Among those approximations are the single scattering assumption and the nonshadowed assumption. A single scattering assumption simply ignores multiple bounces of light, which is accurate if the particles are not too shiny (called low albedo or reflectivity). The nonshadowed assumption is that the light sources are not shadowed as they illuminate the volume, which saves work on calculating the light source contribution.

Another important approximation which saves computation is to classify, shade, and then resample, because the data are often resampled without reclassification. Choices such as these result in different volume rendering pipelines. Figure 1 shows a pipeline that does classification and shading before resampling.

In this paper we assume the optical model of absorption plus emission [Max95], as a solution of the following differential equation (from Max),

$$dI/ds = C(s)\tau(s) - \tau(s)I(s) \qquad (1)$$

where I is the light intensity at distance s, $C(s)$ is the intensity (color) reflected from a point in the volume, $\tau(s)$ is an extinction coefficient. The extinction coefficient can be used to solve for opacity, α, for a given length of material usually by a Taylor series approximation to the exponential:

$$\alpha = 1 - e^{-\int_0^l \tau(t)dt} \cong \tau l - (\tau l)^2/2 + \ldots \qquad (2)$$

So that the solution integral, when evaluated by a Riemann sum is approximated by the following:

$$\tilde{C}_{ray} = \sum_{n=1}^{N} C[n]\alpha[n]\prod_{m=1}^{n-1}(1-\alpha[m]), \qquad (3)$$

for N sample points along a ray, indexed by n, of colors $C[n]$, opacities $\alpha[n]$, and the transparency of the material in front of a ray point computed by the product of the local transparencies, $(1-\alpha[m])$. Such a solution has been used by Blinn and others [Bli82,KH84,DCH88,Lev88,Max95 EQ page 103, Witt93 Eq 61]. Though researchers [Max95,Sab88,WVG91,Wil91] have also mentioned the possibility of rendering a completely transparent glowing gas, we do not address that possibility here. A physically based model, assuming classification determines particle densities and colors is assumed for the analysis here, and would be an appropriate model for medical visualization and volume graphics.

2 SEPARATE INTERPOLATION OF COLOR AND OPACITY

Using the approach shown in Figure 1, the optical model basis just discussed, EQ (3), let us examine in detail how to perform the calculation. We follow Blinn's [Bli94] notation for an opacity-weighted color, which he calls an associated color, and denotes it as $\tilde{C} = \alpha C$ where C is simply the nonweighted color, or gray level. The compositing equations are [Bli94] (for either front-to-back or back-to-front)

$$\tilde{C}_{new} = (1-\alpha_{front})C_{back}\alpha_{back} + C_{front}\alpha_{front} \qquad (4)$$

$$\alpha_{new} = (1-\alpha_{front})\alpha_{back} + \alpha_{front} \qquad (5)$$

When iterating front-to-back we can use

$$\tilde{C}_{front(new)} = (1-\alpha_{front})C_{back}\alpha_{back} + \tilde{C}_{front(old)} \qquad (6)$$

We can also iterate back-to-front with the following equation [Lev88]:

$$\tilde{C}_{back(new)} = (1-\alpha_{front})\tilde{C}_{back(old)} + C_{front}\alpha_{front} \qquad (7)$$

In [Lev88] the ray colors are not described as associated colors, which is correct if an opaque background is used. Figure 2A shows a ray exiting a fully opaque region, into a region of empty space. Here a simplified 2D volume is raycast with a 1D ray. Voxels are classified and shaded before interpolation. Then, interpolation is performed. Three ray samples are shown (bilinear interpolation), with their opacities and colors from front-to-back: $\alpha_1 = 0$, $C_1 = 0$, $\alpha_2 = 0.5$, $C_2 = 0.5$, $\alpha_3 = 1$, $C_3 = 1$. These are the results of the "Ray tracing/resampling" step in Figure 1. Now the three sample points are composited front-to-back, first 1 and 2, where $\tilde{C}_{12}$ is front=1, back=2, substituted into the iterative version of EQ 4, EQ 6, and similarly for α_{12} where we use EQ 5.

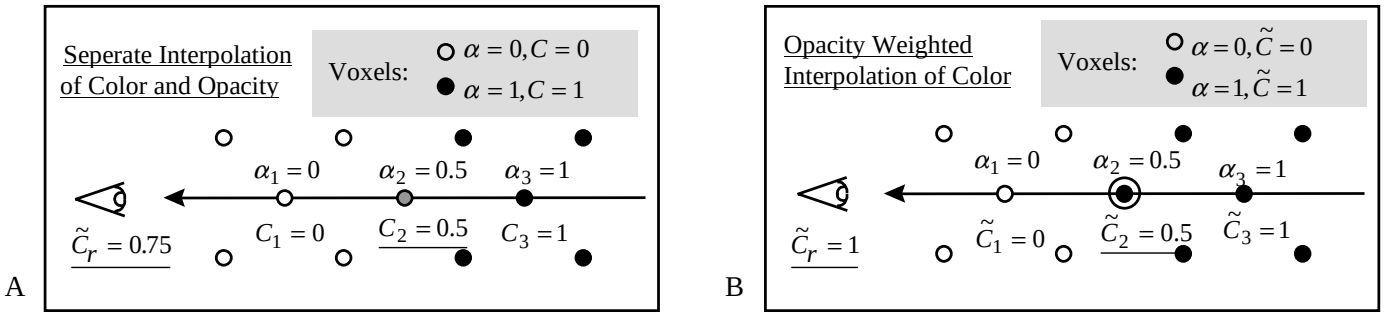

Figure 2. A ray leaving a fully opaque material into empty space showing sample points and compositing results. (A) shows the result with independent interpolation of color, and opacity, as is commonly done. This incorrect result has allowed colors in an empty region of space to contribute to the ray. (B) shows the correct results with opacity-weighted interpolation. See Figure 4 for details on the interpolation.

$$\tilde{C}_{12} = (1-\alpha_1)C_2\alpha_2 + C_1\alpha_1$$
$$= (1-0)\times 0.5\times 0.5 + 0\times 0 = 0.25$$
$$\alpha_{12} = (1-\alpha_1)\alpha_2 + \alpha_1$$
$$= (1-0)\times 0.5 + 0 = 0.5$$

Then (12) are composited with 3:

$$\tilde{C}_{123} = (1-\alpha_{12})C_3\alpha_3 + \tilde{C}_{12}$$
$$= (1-0.5)\times 1\times 1 + 0.25 = 0.75$$
$$\alpha_{123} = (1-\alpha_{12})\alpha_3 + \alpha_{12}$$
$$= (1-0.5)\times 1 + 0.5 = 1$$

The resultant ray color is $C_r = \tilde{C}_{123}/\alpha_{123} = 0.75$. This is in error, and can be shown by a simple example in the next section where we also present our new technique.

3 OPACITY-WEIGHTED COLOR INTERPOLATION

Figure 2B shows the same scenario, but now we shall opacity-weight the colors before interpolation. The opacity-weighted color shall be called an associated color $\tilde{C} = \alpha C$, and shall be explicitly calculated by multiplying all sample points by their opacity. We shall also now use a slightly different compositing equation that is equivalent to EQ 4, but uses associated colors [Lev90,Bli94],

$$\tilde{C}_{\text{new}} = (1-\alpha_{\text{front}})\tilde{C}_{\text{back}} + \tilde{C}_{\text{front}} \quad . \qquad (8)$$

For opacity, we use the same compositing as before, EQ (5). Note the change in the Figure 2B key. Three ray samples are shown (bilinearly interpolated), with their opacities and associated colors from front-to-back: $\alpha_1 = 0$, $\tilde{C}_1 = 0$, $\alpha_2 = 0.5$, $\tilde{C}_2 = 0.5$, $\alpha_3 = 1$, $\tilde{C}_3 = 1$. Figure 3 shows a new volume rendering pipeline, and the ray sample values are the results of the "Ray tracing/resampling" step after the opacity weighting. Now

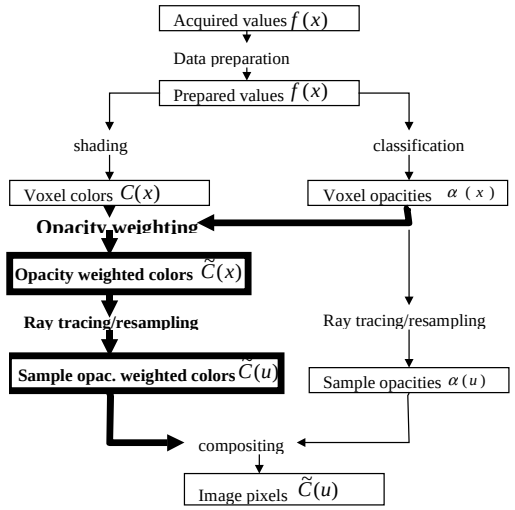

Figure 3. Opacity-weighted color interpolation pipeline analogous to Figure 1. Uses correct opacity-weighted colors for interpolation, with modified and added blocks in bold. Note, that opacity weighting must occur prior to interpolation for the ray resampling, and hence is not possible to be corrected for in the compositing stage following interpolation.

the three sample points are composited front-to-back, first 1 and 2:

$$\tilde{C}_{12} = (1-\alpha_1)\tilde{C}_2 + \tilde{C}_1$$
$$= (1-0)\times 0.5 + 0 = 0.5$$
$$\alpha_{12} = (1-\alpha_1)\alpha_2 + \alpha_1$$
$$= (1-0)\times 0.5 + 0 = 0.5$$

Then (12) are composited with 3:

$$\tilde{C}_{123} = (1-\alpha_{12})\tilde{C}_3 + \tilde{C}_{12}$$
$$= (1-0.5)\times 1 + 0.5 = 1$$
$$\alpha_{123} = (1-\alpha_{12})\alpha_3 + \alpha_{12}$$
$$= (1-0.5)\times 1 + 0.5 = 1$$

The resultant ray color is $C_r = \tilde{C}_{123}/\alpha_{123} = 1$. As we will show, this is the correct result for the optical model shown. The error occurs at the resample point falling between the materials. To summarize, a complete calculation of the resampled ray color at the transition is shown in Figure 4. The color is 1.0 using opacity-weighted color interpolation and 0.5 by separate color and opacity interpolation, which is not consistent with our optical model.

We propose that proper calculation of resampled colors is done by weighting each source voxel color value, (C_r, C_g, C_b) by the opacity α. This is similar to Drebin et al. [DCH88], but has some important advantages that are discussed in Section VII. These weighted colors are then interpolated, as are the opacity values. A ray interpolated sample opacity and color are calculated as:

$$\alpha = \sum_i w_i \alpha_i \qquad (9)$$

$$\tilde{C} = \sum_i w_i \alpha_i C_i = \sum_i w_i \tilde{C}_i \qquad (10)$$

The weights, w_i, are the percentage contribution for each source volume value sample point. For trilinear interpolation there are eight weights. The compositing is then done using EQ (5) and EQ (8) using these opacity-weighted interpolated colors at each ray sample point. At each source voxel we choose to store opacity (α) and unweighted color (C), and perform the opacity weighting during interpolation. For the case where α is 0, the products with (C_r, C_g, C_b) are zero, therefore the colors may be set to 0. Since the products $w_i\alpha_i$ are used for each channel, they can be computed once as $\omega_i = w_i\alpha_i$, resulting in:

$$\alpha = \sum_i \omega_i \qquad (11)$$

$$\tilde{C} = \sum_i \omega_i C_i \qquad (12)$$

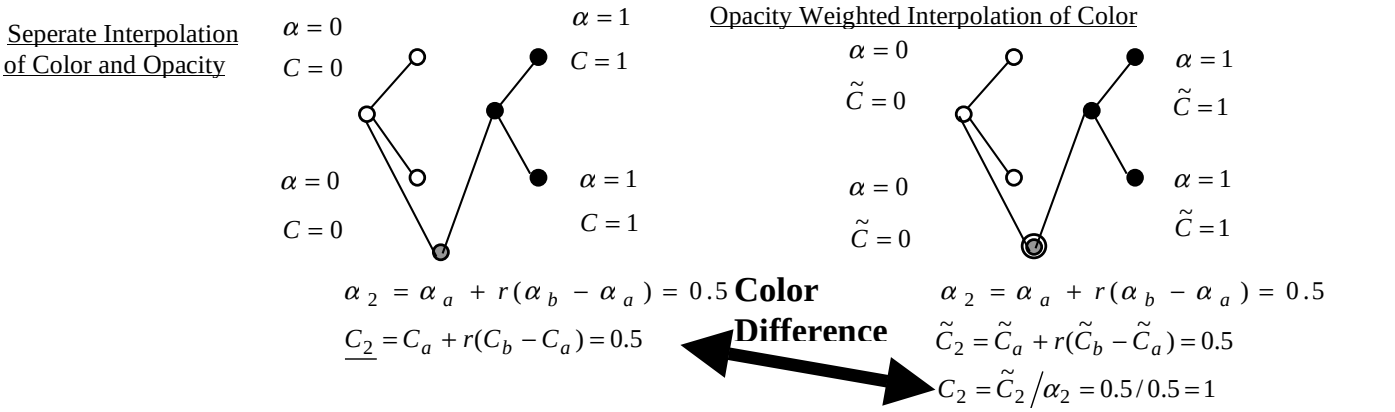

Figure 4 – The determination of color at the boundary between empty space and material, showing a bilinear interpolation as repeated linear interpolations, with the distance r, between the samples as 0.5 in the vertical and horizontal directions. Two different colors result, 0.5 for separate interpolation, and 1.0 for opacity-weighted interpolation. This is the cause of the error.

Our method uses efficient calculation, with minimal storage requirements, and computes correct ray colors. Next we analyze a continuous model of this example.

4 EXAMPLE PROBLEM REVISITED

In the problem demonstrated in Section 2 and Section 3, there was a difference in the colors when interpolating an alpha weighted color versus interpolating a non alpha weighted color. The error shown results from the approximation to shading points along the ray, by preshading and then resampling those sampled colors. The artifact shown above varies with the ray phase, or the distance from the closest sample point to the material. Consider the scenario of driving rays into a material. Figure 5 shows a solid

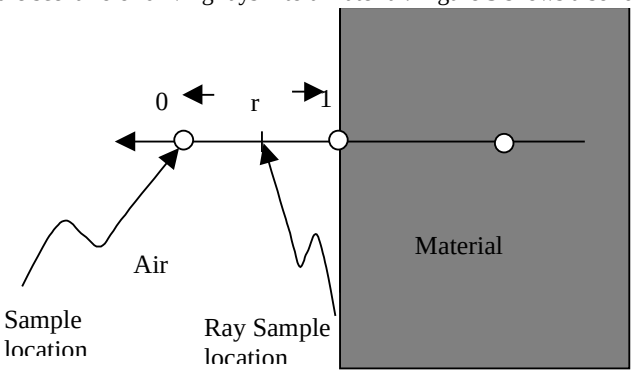

Figure 5. Test scenario of rays entering material from air, with parameter r, for location of sample point between the air sample location and the material sample location.

material with a sharp interface. Consider volume samples to lie within, either the air surrounding the material, or in the material. Define the phase parameter, r, to be the value between 0 and 1, where the ray resampling point lies between the voxel sample point in air, and the voxel sample point in the material. With the color and opacities of the previous example, we can analytically solve for the color for each interpolation approach. For non opacity-weighted color interpolation we can use EQ 4 to composite the results. Since the parameter, r, varies between 0 and 1, and the color and the opacity do as well, a linear interpolation of both the opacity and color will be equal to r. We can substitute into EQ 4,

$$\tilde{C}_r = (1-r)1\times 1 + r\times r = 1 - r + r^2 .$$

Figure 6 gives a plot of the color that results for different phases of the ray. A different color results for every value of r, and the example in Section 2 happened to use the worst case when $r = 0.5$, and the resulting color is $C = 0.75$.

For opacity-weighted interpolation we can solve the same scenario. The interpolated associated color and opacity are equal to r, and we composite with EQ 8,

$$\tilde{C}_r = (1-r)1 + r = 1 - r + r = 1 .$$

The result is a constant color of 1, irrespective of the value of r. The value of the ray color will always be the color of the material. It is straightforward to generalize this to a generic color variable. Consider the boundary between the air and material in Figure 5. For a ray sample rate the same as the volume sample rate, there will always be 1 ray sample in the transition between material and air at the boundary and a sample within the material, as shown before. Interpolated values at the transition will be $r \times$ value_in_the_material. Define the color of the material to be C_m, the opacity of the material to be α_m, then substituting into EQ 4 and EQ 5, and solving for the resultant color, we get for separate interpolation of color and opacity

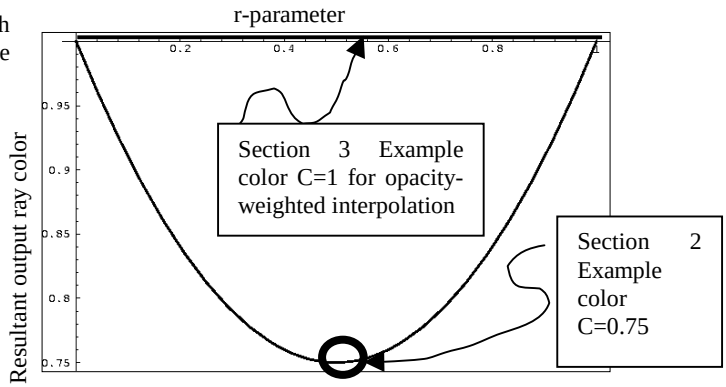

Figure 6. Plot of parameter r versus the resultant ray color for different phase. r represents distance of nearest sample to wall of material. Opacity-weighted color interpolation would give a color of 1 for all values of r.

$$\tilde{C} = (1 - r\alpha_m)C_m\alpha_m + rC_m r\alpha_m = C_m\alpha_m(1 + r^2 - r\alpha_m)$$

For opacity-weighted color interpolation, where interpolated opacity and color are $r\alpha_m$ and $r\tilde{C}_m$, we composite with EQ 5 and EQ 8 to get

$$\tilde{C} = (1 - r\alpha_m)\tilde{C}_m + r\tilde{C}_m = \tilde{C}_m(1 + r - r\alpha_m)$$

If interpolation was done instead on material values, which are then converted to colors, then the opacity-weighted color interpolation is equivalent if the classifier is a linear ramp or a step. Material value, m, is defined as a scalar that is classified to some color and opacity by a function or a lookup table $(C_m, \alpha_m) = \text{Classify}(m)$. This implies that material based interpolation will be equivalent to opacity-weighted color interpolation for a certain class of material classifiers. The interpolated material value would be equal to $r \times m$.

The color will depend on a transfer function. Consider a step function. If the interpolated material is equal to m or more, than $\tilde{C}_m$ results, the same result as opacity-weighted color interpolation for $\alpha_m = 1$. A ramp would be color $r\tilde{C}_m$ at the transition. Input $\tilde{C}_m$ as the back color, $r\tilde{C}_m$ as the front color, and the reduction of the back color is by a factor $(1 - r\alpha_m)$ to give,

$$\tilde{C} = (1 - r\alpha_m)\tilde{C}_m + r\tilde{C}_m = \tilde{C}_m(1 + r - r\alpha_m)$$

This is the exact same as the opacity-weighted color interpolation for arbitrary α_m. Note, that for the general result, an α_m value of 1, the color equation reduces to the material color, but for smaller opacities the color will vary depending on the ray phase. The non opacity-weighted interpolation will vary by a quadratic term. This example provides more intuition as to why opacity-weighted color interpolation is more correct, because it approximates material based interpolation for step and linear material classifiers, and eliminates the dependencies of ray colors on the phase, r, for solid materials.

5 RESULTS

The interpolation of unweighted colors miscalculates the shading of new sample points along rays. In volume rendering such errors can result in color shifting, darkening, greater aliasing, and contribution of color from null regions. We show several examples of artifacts resulting from this error.

Figure 8 shows two volume renderings of a volumetric torus. The dataset is a sampled analytic torus function that has been low pass filtered to 8 bit voxels for antialiasing. This volume is then classified before rendering using a standard isosurface classifier [Lev88]. The rendering itself uses a step size of one half of the voxel spacing. The rendering using separate interpolation of colors and opacity causes severe banding artifacts since the colors assigned to empty space contribute to the ray with this scheme. The rendering using opacity-weighted color interpolation, Figure 8 right, does not show these artifacts.

The next example, shown in Figure 9, compares interpolation strategies on a rendering of a computed tomography dataset. This 8-bit dataset (100 x 96 x 249 voxels) includes several human vertebrae and was collected with spiral CT. Data is courtesy of Dr. Ramani Pichumani, Stanford University. Classification was also performed as a preprocess, and empty space (opacity = 0.0) surrounding the thresholded vertebrae was colored red. This readily labels surface regions where inappropriate colors are being introduced. The data was rendered using volumetric ray-casting with a step size of one half of the voxel spacing. Note that the worst artifacts appear (red areas in left image) when rays are able to graze along the surface for some distance, leading to a result that is dependent on surface orientation relative to the viewer. Animation also shows that the artifact causes banding and is view dependent.

We have additionally investigated the variation of the method with some test objects. Three test objects are shown in Figure 10. The test objects are density volumes, that are classified to a single material, and we call them corner, graze, and phase. These three empirical test cases were designed to further evaluate the analytical investigation of error by phase of the ray. Each volume has been synthesized, some of them with antialiasing along the principal face. The antialiasing was performed by using a windowed sinc.

For the corner scenario, Figures 11 and 12 show the resultant differences between non-opacity-weighted shading, left, and opacity-weighted shading, first from left. The difference is shown in both the same units as the original images, and also in histogram equalized form, so as to make the artifacts obvious. The RMS error between the tests is significant. The perceptual differences vary by scenario, volume resolution, and whether the antialiasing of the source data was performed. Figure 7 shows a plot of RMS error, with the antialiased and aliased volumes split into two rows of bars (AA versus non-AA). The antialiasing reduces the difference in all cases. For experiments we have run, errors are higher for non-antialiased source volumes. Errors are also higher for lower resolution source inputs. Shown are 32^3 and 128^3 source volumes, and the magnitude of the error in rendering the 32^3 source volumes is higher. In all of these test renderings, nonopacity-weighting darkens the results. For bone and white objects a blackened, soot like material appearance results. This is because air is assigned a black color, but it could as easily have been red, as in the case of the vertebrae. In summary, a separate interpolation of color and opacity creates artifacts at the transition

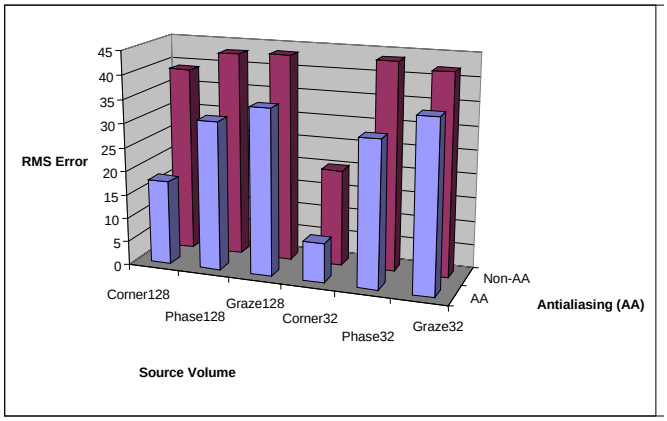

Figure 7. Comparison of RMS Error between non-opacity-weighted shading and opacity-weighted shading for three scenarios (Corner, Phase, Graze), two volume sizes (128^3, 32^3), and using antialiasing or not (AA, Non-AA).

between materials, that may be eliminated or lessened using opacity-weighted color interpolation.

6 HARDWARE IMPLICATIONS

Hardware implications for opacity-weighted color interpolation are an apparent significant increase in work, because of the premultiplication of colors by opacities. We show that this is not the case, and look in more detail at the hardware implementation of EQ's 8, 11, and 12. Interestingly, the cost for performing opacity-weighted color interpolation is the same as nonweighted interpolation if a sum of weighted values are used as shown in EQ's 11 and 12. But, nonweighted interpolation achieves fewer operations when using a binary tree interpolation approach. We have calculated the number of multiplications (M) and additions (A) for four approaches:

Interpolation method	Binary tree	Sum of weighted products
Separate color/opacity	28M, 56A	36M, 47A
Opacity-weighted color	52M, 56A	36M, 47A

Table 1. Multiplication (M) and addition (A) costs for separate color/opacity interpolation and opacity-weighted color interpolation.

These costs are for the interpolation of an RGBα value for a sample point. The compositing costs turn out to be the same for either approach even though different equations are used. The binary tree interpolation is often used for trilinear interpolation in volume rendering, and involves repeated linear interpolations. The method is shown in Figure 4 for bilinear interpolation. The computation can perform, for a single trilinear interpolation of a value, four interpolations in x, two interpolations in y, and one final interpolation in z. If the four corners of an area between point samples are labeled with 0 or 1 depending on their range in dimensions X, Y, Z, the interpolation would be calculated as:

$$f(x,0,0) = f[0,0,0] + x[f[1,0,0] - f[0,0,0]]$$
$$f(x,1,0) = f[0,1,0] + x[f[1,1,0] - f[0,1,0]]$$
$$f(x,0,1) = f[0,0,1] + x[f[1,0,1] - f[0,0,1]]$$
$$f(x,1,1) = f[0,1,1] + x[f[1,1,1] - f[0,1,1]]$$

Then in dimension y using values from dimension x:

$$f(x,y,0) = f(x,0,0) + y[f(x,1,0) - f(x,0,0)]$$
$$f(x,y,1) = f(x,0,1) + y[f(x,1,1) - f(x,0,1)] .$$

And finally in dimension z:

$$f(x,y,z) = f(x,y,0) + z[f(x,y,1) - f(x,y,0)] .$$

Where (x,y,z) is the fractional domain location value where the interpolation is to be performed, and is assumed to range between 0 and 1 inside the 8 values being interpolated. Brackets are used for array values [], and parenthesis for continuous functions ().

To compute the sum of weighted values approach, requires computing eight weights for each sample point to contribute to the final interpolated voxel, w_i. Then the samples are all multiplied and summed as shown in EQ 12. To calculate the weights, the following equations are used:

$$w[0,0,0] = (1-x)(1-y)(1-z)$$
$$w[0,0,1] = (1-x)(1-y)(z)$$
$$w[0,1,0] = (1-x)(y)(1-z)$$
$$w[0,1,1] = (1-x)(y)(z)$$
$$w[1,0,0] = (x)(1-y)(1-z)$$
$$w[1,0,1] = (x)(1-y)(z)$$
$$w[1,1,0] = (x)(y)(1-z)$$
$$w[1,1,1] = (x)(y)(z)$$

These weight value equations may also be expanded to be computed with fewer multiplications, or computed as shown to reduce the latency in the circuit necessary to compute them.

Considering binary tree interpolation, for the nonweighted interpolation, simply calculate four trilinear interpolations, one for each RGBα. Each trilinear interpolation is decomposed into seven linear interpolations, which is most efficiently calculated as one multiply with two additions, so 4x7x(1M,2A) equals 28M, 56A. When performing opacity-weighted color interpolation, there is the added cost of 3x8 multiplies to opacity weight the colors, totaling 52M, and 56A.

The sum of weighted values approach, for the nonweighted interpolation, computes the weights with 4 multiplies, 19 additions, and then computes four sum-of-weighted-calculations, requiring eight multiplies and seven additions each (32M, 28A). The total is 36 multiplies and 47 additions. For weighted interpolation, because the ω values are computed, the weighting is not done redundantly, so we simply compute the weights (4M, 19A), premultiply weights (8M), sum for alpha (7A), and perform 3 weighted interpolations 3x(8M,7A), which totals 36M and 47A the same cost as nonweighted interpolation. Calculating conditionals on voxel values allows implementations to skip interpolations if voxels, or interpolated opacity equals zero.

What this analysis shows is that it is possible for only a small additional cost 36M versus 28M to do opacity-weighted color interpolation, and additionally, the proper way to most efficiently compute that is by sum of weighted interpolated values.

7 RELATED WORK

The popularity of ray casting [Lev88,Lev90] has lead to the interpolation artifact appearing in a number of volume rendering implementations. Opacity-weighted shading is a small improvement, but is also easily misunderstood. The pervasiveness of the artifact is also hard to determine. But, opacity-weighted color interpolation is an unpublished technique as demonstrated here except for Drebin et al. [DCH88] who preweight entire volumes. Examination of source code has shown others [LL94] have used similar solutions.

Drebin et al. [DCH88] specify opacity weighting of the colors for volume rendering by preweighting. Others have noted the difficulties associated with opacity and color, notably Wilhelms and Van Gelder [WVG91] and Blinn [Bli94], but not as it relates to resampling of shaded color values in volume ray tracing. Wilhelms notes that different answers will result if one interpolates colors versus data, but the former may be faster. We have actually shown here that interpolating opacity-weighted color can provide the same answer in some circumstances. Blinn discusses the issues in filtering an image that has opacities at each pixel. He shows an example of overlaying and downsampling, and the different answer that results if downsampling first without opacity weighting the colors. He calls the opacity-weighted colors

associated colors in perhaps the clearest and most concise discussion of this subject. Wilhelms, Van Gelder et al. [Wil91,VGK96] discuss the issues in interpolating colors instead of materials, and resulting artifacts, but do not clarify that additional artifacts result if interpolated colors are not weighted by opacity.

We demonstrated the problem encountered by separate interpolation of color and opacity in the context of ray cast volume rendering in this paper. However, the suitability of the interpolation method presented is not limited to ray-casting and applies to other techniques such as splatting [Wes90], three-dimensional texture mapping [VGK96], and polygon rendering as well.

There are three alternative approaches to avoiding the interpolation artifacts described in Section 2. These are summarized in Table 2. The first alternative is to store raw data (material) values themselves, interpolate these down to ray samples, and then classify the interpolated results [Osb97,Sch96,Avi94,Hoh90]. Sub-voxel surfaces may be reconstructed by this approach, but there are drawbacks. First, classification *must* be performed for every frame being computed, unlike the preclassification case. Computational cost is also typically higher since there are often more ray sample points than voxels. Second, since classification acts on sample points whose positions are view dependent, material interpolation can introduce view dependent.

The second alternative is to store classified-shaded-color values preweighted by opacity, along with opacity at each voxel, as was done in Drebin et al. [DCH88]. Interpolation is then performed on preweighted colors. Preweighted colors, however, require more bits of precision than unweighted colors for the same image quality. This is because opacity weighting returns a fraction of the shaded values, which if rounded to be stored in fixed point will reduce the accuracy of subsequent calculations. Rounding a floating-point number to a fixed-point representation results in a maximum error of 0.5 But, when opacity-weighting, we are rescaling the values so the color error is multiplied. To convert to fixed point, a number is scaled, then rounded. A floating-point color, C, in the range of $[0,1]$, can be converted to fixed point by $C_F = \text{Int}(C \times (N-1) + 0.5)$, for N levels in the range $[0, N-1]$ and the integer operator $\text{Int}()$. The floating point scaled color value, we call C_F, and the fixed-point value is $C'_F = C_F + \varepsilon_C$, where $|\varepsilon_C| \le 0.5$. Now we can scale and convert an associated color in the same way $\tilde{C}'_F = \tilde{C}_F + \varepsilon_C$. But, the error in the unweighted color must be calculated by dividing out the opacity. The exact opacity is $\alpha = \alpha_F / (N-1)$, considering no error to represent α_F, the fixed-point opacity. The 0.5 bit error is scaled when recalculating the unweighted color as $C'_F = \tilde{C}'_F / \alpha = \tilde{C}'_F / (\alpha_F / N-1)$. Which is equal to $C'_F = C_F + \varepsilon_{\tilde{C}} (N-1 / \alpha_F)$, by substitution and rearranging of terms. The rounding error in the associated color, $\varepsilon_{\tilde{C}}$, is scaled by $(N-1)/\alpha_F$. As an example consider rounding 8 bit colors and opacities, $N = 256$, and $N-1 = 255$. The error will be the maximum of $0.5(255/\alpha_F)$ for $\alpha_F \in [1,255]$. The smallest opacities create the largest errors say that $\alpha_F = 1$ results in an error $0.5(255/1) = 127$ for values ranging from 0 to 255, or half of the range of the color (half of the bits). Errors will be large when rendering mostly transparent materials.

In the proposed alternative a higher precision representation can be used during weighting which results in greater accuracy. Three-dimensional texture mapping using hardware acceleration [VGK96] does not opacity weight colors, but it could be done by preweighting which may result in difficulties due to the fixed point texture storage formats.

The third alternative is opacity-weighted color interpolation as described here. Color and opacity values are stored independently. Opacity-weighting is performed during interpolation for memory savings (bit precision), computational advantages of preclassification, and comparable speed to preweighting colors. For example 8 bit colors can be loaded into 32 bit registers, and then alpha weighting may be performed on the higher precision representation.

	Method	**Advantages**	**Disadvantages**
(1)	Interpolate then classify	Sharper surface detail compared to (2,3)	Classification: *view dependent *needed per view
(2)	Store opacity-weighted colors	shorter render time compared to (1)	Requires more bits/voxel than (3)
(3)	Store Unweighted Colors, α weighted interpolation	short render time less bits/voxel	Softer imagery compared to (1)

Table 2 - Volume rendering approaches that avoid this interpolation artifact.

8 CONCLUSIONS

We show an artifact in volume rendering software and algorithms. We show the origins of the artifact in the literature, and identify the cause of the artifact— an improper interpolation method. The artifact appears as artificial darkening in monochrome images, highlighting of aliasing errors, or as color shifts in RGB rendering, actually creating misleading renderings. We clarify that the situation occurs only when classifying and shading before resampling, and present a solution for this case. We refer to this solution as ***opacity-weighted color interpolation***. Prior published solutions have either interpolated ray values and then classified and shaded, stored entire opacity-weighted volumes which requires more storage precision, or interpolated incorrectly.

The issue of whether this artifact is widely seen or embedded in production code is difficult to ascertain. Other implementations get around these difficulties, primarily by doing interpolation of materials [Osb97,Sch96,Avi94,Hoh90]. We have determined that preweighting has been used in some implementations. Drebin et al. [DCH88] correctly discuss the opacity color issues, and actually use preweighted associated color volumes. But, because of the scaling issues, this achieves a less accurate rendering with a fixed-point format than our method.

Lacroute and Levoy [LL94] do properly opacity weight as determined by examining the publicly available code. They premultiply during the time that voxel color values are loaded, and that code base is widely used. But the advantages and elimination of artifacts by doing this have not been published. The preclassified color method [Lev88,Lev90] describes the separate interpolation of color and opacity values. The widest use of the improper technique seems to be implementations that use

3D texture mapping, as the color values are not preweighted, and are not weighted prior to interpolation (such as [VGK96]).

The discovery of a method to directly render from preshaded values makes software and hardware more efficient Opacity-weighted color interpolation is an important clarification to the technique. It allows for a more accurate solution with the same efficiency. We found error magnitudes are data set, view, classification, and shading dependent, which may account for the oversight. Also, because of the number of steps in volume rendering, other artifacts may dominate such as reconstruction error, gradient calculation, and shading, reducing the relative importance of opacity weighting. In fact, the same artifact as shown for volume rendering is important for polygon graphics. We believe this work represents an easy fix for artifacts that exists in all texture mapping implementations that use color and opacity textures. Further work is needed to quantify the amount of error that results, but texture mapping could be additionally improved with opacity-weighted color interpolation. While the misuse of opacity and color in volume rendering is subtle, we believe that this work clarifies an important nuance in a widely used algorithm, and will improve the quality and understanding of future implementations having an impact for nearly all computer graphics.

Acknowledgments

We would like to thank Kevin Wu who helped with the animations, and the reviewers who have helped to tremendously improve the presentation.

References

[Avi94] R. Avila et al. "VolVis: a diversified volume visualization system," In *Proc. Of Visualization*, pages 31-38, IEEE, Washington, D.C. Oct. 1994.

[Bli82] J. Blinn, Light reflection functions for simulations of clouds and dusty surfaces. In *Proc. of SIGGRAPH*, pages 21-29, ACM, New York, NY, July 1982.

[Bli94] J. Blinn, Jim Blinn's corner: Compositing, part I: Theory. *IEEE Computer Graphics and Applications*, pages 83-87, Sep. 1994.

[Cha60] S. Chandrasekhar. Radiative Transfer Theory. Dover, New York, NY, 1960.

[DCH88] R. A. Drebin, L. Carpenter, and P. Hanrahan, Volume rendering In *Proc. of SIGGRAPH*, pages 65-74, ACM, New York, NY, August 1988.

[Esp79] Larry W. Esposito, Extensions to the classical calculation of the effect of mutual shadowing in diffuse reflection, Icarus, 39:69-80, 1979.

[Hoh90] K.H. Hohne et al., "A 'virtual body' model for surgical education and rehearsal", *IEEE Computer*, Vol. 29, No. 1, pages 25-31, Jan 1996.

[Kau91] A. Kaufman, editor, Volume Visualization. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[KH84] J. T. Kajiya and B. Von Herzen, Ray tracing volume densities, In *Proc. of SIGGRAPH*, pages 165-174, ACM, New York, NY, July 1984.

[KK89] J. T. Kajiya and Timothy L. Kay, Rendering fur with three dimensional textures, In *Proc. of SIGGRAPH*, pages 271-280, ACM, New York, NY, July 1989.

[Lev88] M. Levoy, Display of surfaces from volume data, *IEEE Computer Graphics and Applications*, 8(5):29-37, May 1988.

[Lev90] M. Levoy, Efficient ray tracing of volume data, *ACM Transactions on Graphics*, ACM, New York, NY, 9(3):245-261, July 1990.

[LL94] P. Lacroute and M. Levoy, Fast volume rendering using a shear-warp factorization of the viewing transformation, In *Proc. of SIGGRAPH*, pages 451-458, ACM, New York, NY, July 1994.

[Max86] Nelson L. Max, Light diffusion through clouds and haze, Computer Vision, Graphics, and Image Processing, 33(3):280-292, 1986.

[Max95] N. Max, Optical models for direct volume rendering, *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99-108, June 1995.

[Mea82] D. Meagher, Geometric modeling using octree encoding, Computer Graphics, and Image Processing, 19(2):129-147, June 1982.

[Mea85] D. Meagher. Applying solids processing to medical planning, In *Proceedings of NCGS'85*, pages 372-378, Dallas, TX, February 1985. NCGS.

[Osb94] R. Osborne et al. "EM-Cube: an architecture for low-cost real-time volume rendering," *In Proc. Of Eurographics Hardware Rendering Workshop*, pages 131-138, ACM, Las Angeles, CA, Aug. 1997.

[PD84] T. Porter and T. Duff., Compositing digital images, In *Proc. of SIGGRAPH*, pages 253-259, ACM, New York, NY, Aug. 1984.

[Sab88] P. Sabella, A rendering algorithm for visualizing 3D scalar fields, In *Proc. of SIGGRAPH*, pages 51-58, ACM, New York, NY, Aug. 1988.

[UK88] C. Upson and M. Keeler, V-buffer: Visible volume rendering, In *Proc. of SIGGRAPH*, volume 22, pages 59-64, ACM, New York, NY, July 1988.

[VGK96] A. Van Gelder and K. Kim, Direct volume rendering with shading via 3D textures, *In Symposium on Volume Visualization*, pages 23-30, San Francisco, CA, Oct. 1996.

[Wes90] L. Westover: Footprint Evaluation for Volume Rendering, in *Proc. of SIGGRAPH*, volume 24, no. 4, pages 367-376, ACM, New York, NY, July 1990.

[WVG91] J. Wilhelms and A. Van Gelder, A coherent projection approach for direct volume rendering, In *Proc. of SIGGRAPH*, pages 275-284, ACM, New York, NY, 1991.

[Wil91] J. Wilhelms, Decisions in Volume Rendering, In *SIGGRAPH 91 Course Notes 8-State of the Art in Volume Visualization*, pages I.1-I.11, ACM, New York, NY, 1991.

[Witt93] C. M. Wittenbrink, Designing optimal parallel volume rendering algorithms. Ph.D. thesis, University of Washington, 1993.

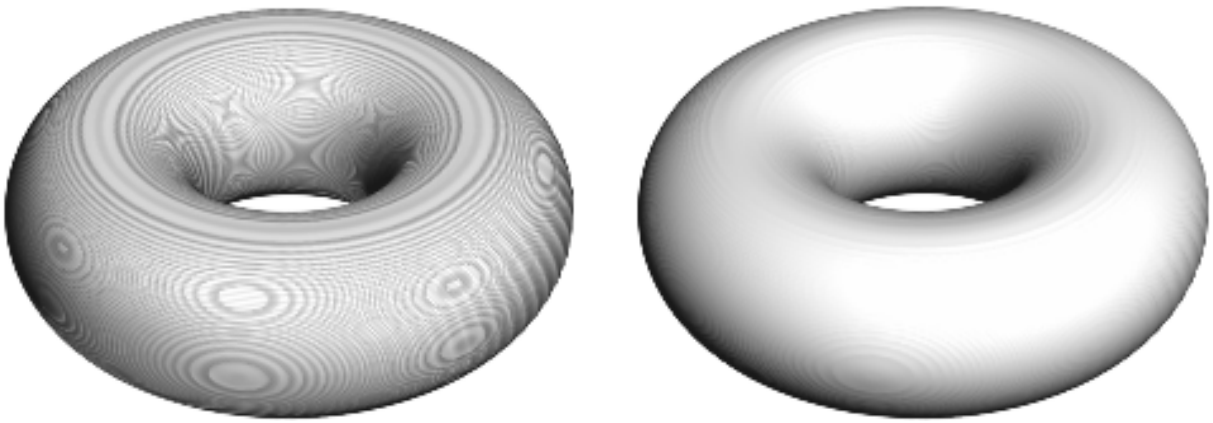

Figure 8. Left: Artifacts of separate interpolation of colors and opacity. Right: Improved using opacity-weighted color interpolation.

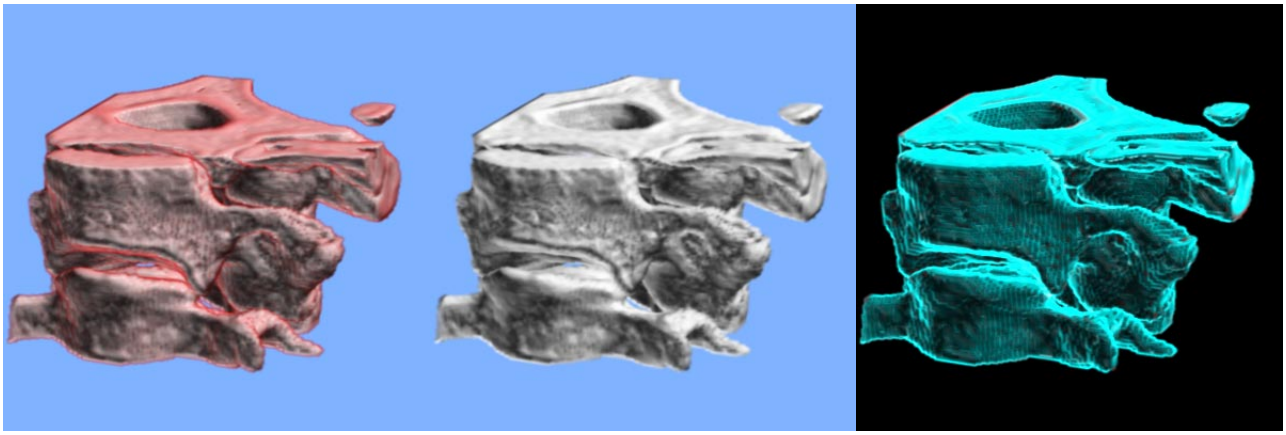

Figure 9. Left: Separate interpolation of color and opacity. Middle: Opacity-weighted interpolation of colors. Right: Normalized difference Image. Data courtesy of Dr. Ramani Pichumani, Stanford University.

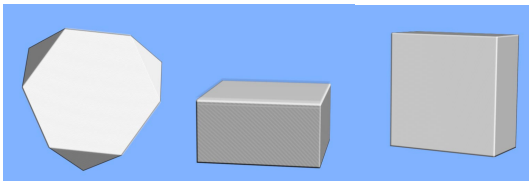

Figure 10. Three test rendering scenarios: corner, graze; and phase. One plane of the material boundaries has been antialiased during formation, by using a windowed sinc interpolation.

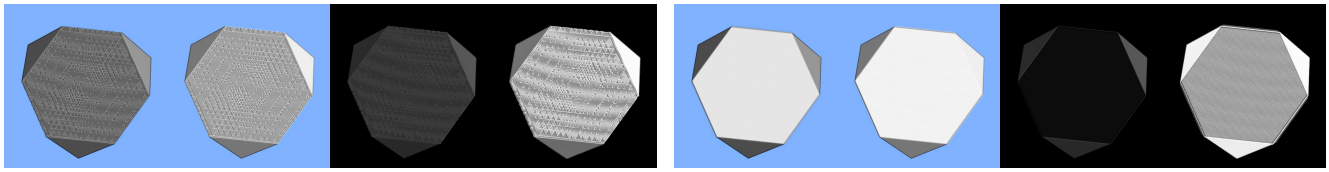

Figure 11. Left four images: from left, i) Artifacts of separate interpolation of colors, ii) opacity-weighted, iii) difference, iv) equalized difference. RMS Error 39.0. Right four images: same view and differences of antialiased face. RMS Error 17.7.

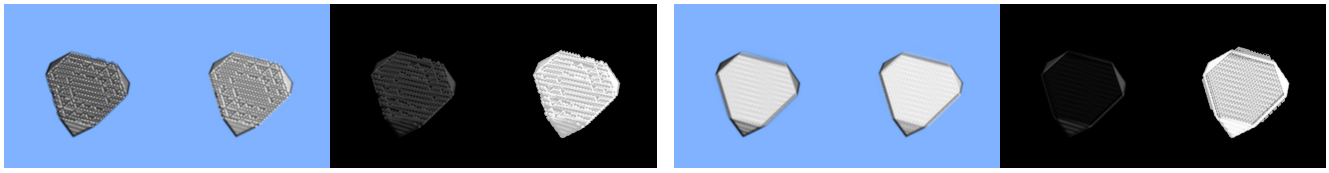

Figure 12. Left four images: from left, i) Artifacts of separate interpolation of colors, ii) opacity-weighted, iii) difference, iv) equalized difference. RMS Error 20.3. Right four images: same view and differences of antialiased face. RMS Error 8.32