Splatting Errors and Antialiasing

Klaus Mueller, *Student Member*, *IEEE*, Torsten Möller, *Student Member*, *IEEE*, J. Edward Swan II, *Member*, *IEEE*, Roger Crawfis, *Member*, *IEEE Computer Society*, Naeem Shareef, and Roni Yagel, *Member*, *IEEE*

Abstract—This paper describes three new results for volume rendering algorithms utilizing splatting. First, an antialiasing extension to the basic splatting algorithm is introduced that mitigates the spatial aliasing for high-resolution volumes. Aliasing can be severe for high-resolution volumes or volumes where a high depth of field leads to converging samples along the perspective axis. Next, an analysis of the common approximation errors in the splatting process for perspective viewing is presented. In this context, we give different implementations, distinguished by efficiency and accuracy, for adding the splat contributions to the image plane. We then present new results in controlling the splatting errors and also show their behavior in the framework of our new antialiasing technique. Finally, current work in progress on extensions to splatting for temporal antialiasing is demonstrated. Here, we present a simple but highly effective scheme for adding motion blur to fast moving volumes.

Index Terms—Volume rendering, splatting, direct volume rendering, resampling, reconstruction, antialiasing, perspective projection, motion blur.

1 Introduction

In the past several years, direct volume rendering has emerged as an important technology in the fields of computer graphics and scientific visualization. Splatting is one of several popular direct volume rendering techniques. Perspective projection introduces the problem of nonuniform sampling produced by diverging viewing rays which, if not properly addressed, can result in potentially severe aliasing artifacts. Although other volume rendering approaches have dealt with this problem (e.g., ray-casting [3], [9] and shear-warp [4]), to date, this problem has not been addressed in the context of splatting. In this paper, we present an antialiasing modification to the splatting algorithm which prevents the aliasing that arises from this non-uniform resampling. The same type of resampling problems also occur with an orthographic projection if the volume resolution is higher than the image resolution (e.g., if many voxels project into each pixel).

Our antialiasing technique was recently presented at the Visualization '97 Conference [11]. The current paper is an expanded version of this conference publication. In addition to a concise presentation of the antialiasing technique itself, the current paper gives a much more detailed treatment of the approximation errors for perspective viewing which are common to our, as well as previous, splatting implementations. It presents both ideal as well as practical solutions to the approximation errors, and analyzes the trade-offs between them in the context of our antialiasing technique. In addition, the paper presents some promising initial results in extending splatting to add motion blur to fast moving volumes.

In the next section, we describe the splatting algorithm and related previous work. In Section 3, we describe our antialiasing technique and present some results. Then, in Section 4, we analyze perspective splatting errors both in the context of traditional splatting and our new antialiased technique. Temporal antialiasing is briefly addressed in Section 5.

2 Previous Work

The splatting technique has been used to directly render volumes of various grid structures [6], [13] and for both scalar [5], [13], [14], [15] and vector fields [1]. The basic algorithm, first described by Westover [13], projects each voxel to the screen and composites it into an accumulating image. It solves the hidden surface problem by using a painter's algorithm: It visits the voxels in either a back-to-front or front-to-back order, with closer voxels overwriting farther voxels.

As each voxel is projected onto the image plane, the voxel's energy is spread over the image raster using an image kernel centered at the voxel's projection point. This image kernel, called a "splat" or "footprint" function, contains the integration of a 3D reconstruction kernel along parallel rays. For spherically symmetric 3D kernels, the direction of the rays is immaterial, and the integration can be precomputed, storing the footprint function in a 2D lookup table. This table is then centered at each voxel's projection point and resampled by the pixels which lie within its extent. As Westover points out [13], [14], [15], for parallel projections and regular voxel grids, the footprint table can be computed

• *K. Mueller is with the Department of Computer and Information Science, The Ohio State University, 2015 Neil Ave., Columbus, OH 43210-1277. E-mail: mueller@cis.ohio-state.edu.*
• *T. Möller, R. Crawfis, N. Shareef, and R. Yagel are with the Department of Computer and Information Science and the Advanced Computing Center for the Arts and Design, The Ohio State University, 2015 Neil Ave., Columbus, OH 43210-1277. E-mail: {moeller, crawfis, shareef, yagel}@cis.ohio-state.edu.*
• *J.E. Swan II is with the Naval Research Laboratory, Code 5580, 4555 Overlook Ave. SW, Washington, DC 20375-5320. E-mail: swan@acm.org.*

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 106603.

once and used for all voxels. For radially symmetric reconstruction kernels, this can also be done once for all views. For perspective views, diverging rays from the eyepoint should be used for the integration. This, then, requires a different footprint function for different voxels. This can be excessively expensive and, to date, the authors know of no implementations that actually perform this.

To improve image quality, in later works, Westover [14], [15] first accumulates splats onto several 2D sheets that are aligned with the volume axis most parallel to the view plane and then composites the sheets in depth order into the image with a matting operation. Image quality is also affected by the size, shape, and type of the reconstruction kernel used. Mueller and Yagel [8] use an image-order splatting approach, which improves accuracy when using a perspective projection. And while, to date, most splatting implementations have used a Gaussian reconstruction kernel, other kernels can generate higher-quality images. Crawfis and Max [1] developed a cubic spline function, optimized for the smooth reconstruction of a constant scalar field.

To improve rendering speed, Westover [14] maps view-dependent footprints with a circular or elliptical shape to a generic footprint table which only needs to be computed once. Laur and Hanrahan [5] approximate splats with a triangle mesh and use graphics hardware to quickly scan convert the footprint. Crawfis and Max [1] and Yagel et al. [16] both use texture mapping hardware to quickly render the splats. In this technique, the footprint is texture mapped onto polygons located at the voxel positions. Splatting can also be accelerated by preprocessing the volume and culling voxels which will not contribute to the final image. Laur and Hanrahan [5] cull with an octree structure, and Yagel et al. [16] extract and store only the most visually significant voxels.

This paper focuses on improved image quality for splatting, while striving to maintain optimal performance.

3 An Antialiasing Technique for Splatting

In this section, we describe an antialiasing technique for splatting. In Section 3.1, we discuss sources of aliasing and place them in the context of volume rendering. In Section 3.2, we develop an expression (2) which, if satisfied by a given volume-rendering algorithm, indicates that the algorithm will not produce the sample-rate aliasing artifacts that arise from the resampling phase of the rendering process. In Section 3.3, we describe our antialiasing method, and, in Section 3.4, we show that our method satisfies the equation developed in Section 3.2, which argues for the correctness of the method.

3.1 Aliasing in the Volume Rendering Process

The process of volume rendering is based on the integration (or composition), along an *integration grid*, of a density volume. This integration grid is composed of *sight projectors* (or *rays*) which pass from the eye point, through the view plane, into the volume. Before this integration can occur, the density volume usually has to be reconstructed from a discrete voxel grid, and then resampled into the integration grid. This is illustrated in Fig. 1 for a perspective view of the volume, where the volume raster is shown as a lattice of dots, and the integration grid is shown as a series of rays, cast through

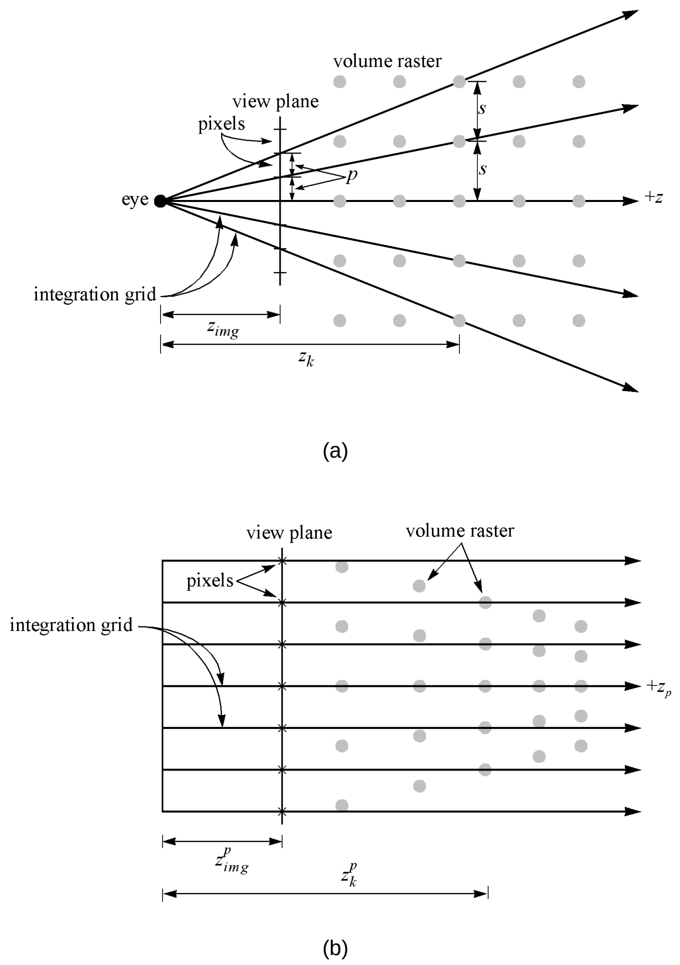


Fig. 1. Resampling the volume raster onto the integration grid. (a) In eye space. (b) In perspective space.

pixels, which traverse the volume raster. Fig. 1a shows the scene in *eye space*, where the eye is located at point (0, 0, 0) and is looking down the positive z-axis (denoted +*z*). For perspective transformations, the integration grid diverges as it traverses the volume. In Fig. 1b, the volume raster is distorted according to the perspective transformation and the integration grid lines are parallel.

The reconstruction of the volume density function and its subsequent resampling by the integration grid is subject to aliasing if the resampling is below the Nyquist limit of the reconstructed volume density. If it is not possible to sample above the volume's Nyquist limit, then aliasing can be reduced by low-pass filtering the volume. For an orthographic view, this low-pass filtering is applied uniformly to the entire volume. For a perspective view, the amount of low-pass filtering is dependent on location.

Due to the perspective distortion (Fig. 1b), the frequency content of the density volume may increase (with respect to the image plane) with increasing distance from the eye point. This leads to a different frequency distribution throughout the volume. If we consider a constant z-plane in perspective space, we can determine a proper resampling rate (above the Nyquist limit) for that plane. Furthermore, note that, at the distance z^p_k along the z-axis, the sampling rate of the volume raster and the sampling rate of the inte-

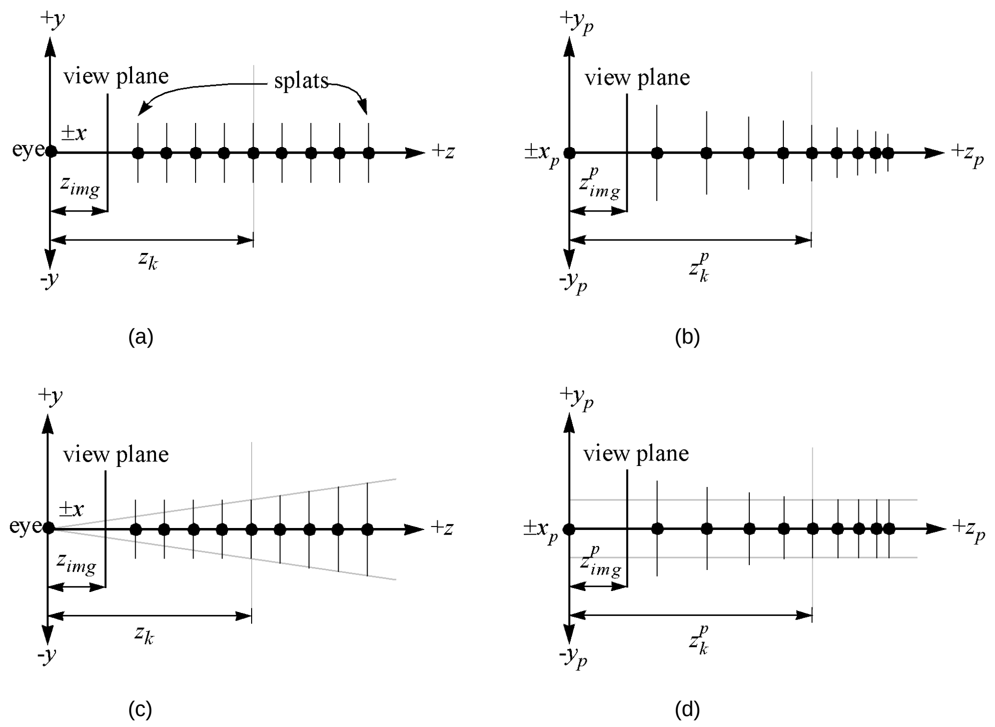


Fig. 2. A comparison of the standard splatting method ((a) and (b)) with our antialiased method ((c) and (d)). (a) The standard splatting method in eye space. (b) The standard splatting method in perspective space. (c) The antialiased splatting method in eye space. (d) The antialiased splatting method in perspective space.

gration grid are the same. Distances closer to the eye point project less than one voxel per pixel and, beyond this distance, there is more than one voxel per pixel. The latter case can contain frequency information higher than the integration grid can represent and, so, aliasing is possible.

3.2 Avoiding Aliasing in Splatting

In this section, we give the conditions which are necessary for the resampling process in eye space to avoid introducing sample-rate aliasing artifacts into the integration grid samples. Let s be the volume raster grid spacing (Fig. 1a), and $\rho = 1/s$ be the volume raster sampling rate. We assume the voxel grid we have been given has been properly filtered and thus

$$\rho \geq 2\omega_{\max}. \tag{1}$$

where $\omega_{\max}$ is the highest frequency of our given density function. Our goal, then, is to avoid aliasing in the resampling process of the integration grid.

Let ϕ represent the sampling frequency of the integration grid. For an orthographic projection, we simply need to ensure that $\phi \geq \rho$. If this is not the case, then a low-pass filter is necessary to lower the value of ρ. For perspective projections, however, the integration grid diverges (Fig. 1a) and ϕ is a function of the distance along the axis from the eye point: $\phi = \phi(z)$. As illustrated in Fig. 1, at a distance z_k, the sampling rates of the volume raster and the integration grid are the same: $\rho = \phi(z_k)$. This distance z_k naturally divides eye space (and, likewise, z_k^p divides perspective space) into the following two regions:

Region 1: $z < z_k$. Here, $\phi(z) > \rho$ or $\phi(z) > 2\omega_{\max}$ and, hence, there is no sample-rate aliasing.

Region 2: $z \geq z_k$. Here, $\phi(z) \leq \rho$ or $\phi(z) < 2\omega_{\max}$ and the integration grid may contain a sample-rate aliased signal.

This argument shows that, beyond z_k, it may be necessary to low-pass filter the volume raster to avoid aliasing. The amount of filtering required increases as a function of z. This reduces the high frequencies in the volume raster from $\omega_{\max}$ to $\tilde{\omega}_{\max}(z)$ yielding the critical condition

$$\phi(z) \geq 2\tilde{\omega}_{\max}(z) \tag{2}$$

for Region 2.

3.3 An Antialiasing Method for Splatting

The distance z_k at which the volume raster sampling frequency ρ and the integration grid sampling frequency $\phi(z)$ are equal can be calculated from similar triangles (Fig. 1a):

$$z_k = s\frac{z_{img}}{p}, \tag{3}$$

where p is the extent of a pixel, and z_{img} is the distance from the eye point to the view plane.

Fig. 2 gives a side view of "standard" splatting, as well as the new antialiasing method. In Fig. 2, the y-axis is drawn vertically, the z-axis is drawn horizontally, the x-axis comes out of and goes into the page, and diagrams are shown in both eye space (x, y, z) and perspective space (x_p, y_p, z_p). The top row illustrates standard splatting. For simplicity, let us assume for now that all splats are aligned parallel to the x-y plane (we will later see (in Section 4) that this is not always the case). In Fig. 2a, a single row of splats

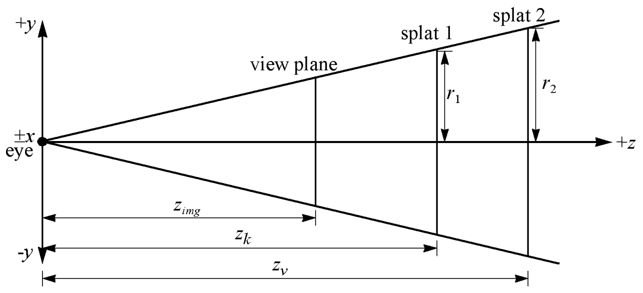


Fig. 3. The geometry for scaling the splats that are drawn beyond z_k.

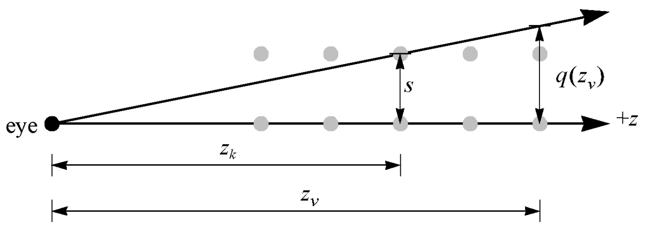


Fig. 4. Calculating the integration grid sampling frequency.

equally spaced along the z-axis is shown. Each splat is considered to be a 2D "footprint" function of limited extent; when seen from the side the splats appear as line segments. Note that each splat is the same size in eye space. Fig. 2b shows the same scene in perspective space. As expected, because of the nonlinear perspective transformation, the splat spacing is now nonuniform along the z_p axis, and the size of the splats decreases with increasing distance from the eye.

The bottom row illustrates our antialiasing method. Previous to z_k, we draw splats the same size in eye space (Fig. 2c). Beginning at z_k, we scale the splats so they become larger with increasing distance from the view plane. This scaling is proportional to the viewing frustum, and is given in (4) below. Fig. 2d illustrates this in perspective space. Previous to z_k^p, we draw the splats with decreasing sizes according to the perspective transformation. Beginning at z_k^p, all the splats are drawn with the same extent.

Fig. 3 gives the geometry for scaling splats drawn beyond z_k. If a splat drawn at distance $z = z_k$ has the radius r_1, then the radius r_2 of a splat drawn at distance $z_v > z_k$ is the projection of r_1 along the viewing frustum. This is calculated by similar triangles:

$$r_2 = \frac{z_v}{z_k} r_1. \tag{4}$$

Scaling of the filter function h implies that its Fourier spectrum H gets scaled and attenuated [10]:

$$h\left(\frac{z_k}{z_v} y\right) \leftrightarrow \left|\frac{z_v}{z_k}\right| H\left(\frac{z_v}{z_k} \omega\right), \tag{5}$$

where "$\leftrightarrow$" indicates a Fourier transform pair. Since we want to scale the frequencies but preserve their amplitude, we must attenuate the amplitude of the splats in world space accordingly, i.e., by a factor of z_k/z_v. If we scale in both the x and y directions, the total attenuation factor is $(z_k/z_v)^2$. This preserves the frequency amplitude:

$$\left(\frac{z_k}{z_v}\right)^2 h\left(\frac{z_k}{z_v} x, \frac{z_k}{z_v} y\right) \leftrightarrow H\left(\frac{z_v}{z_k} u, \frac{z_v}{z_k} v\right). \tag{6}$$

3.4 Correctness of the Method

We now demonstrate that (2) holds for our antialiasing technique. We begin by deriving expressions for the two functions in (2) —$\phi(z_v)$ (the integration grid sampling frequency at z_v) and $\tilde{\omega}(z_v)$ (the maximum volume raster frequency at z_v).

We derive the integration grid sampling frequency $\phi(z_v)$ with a similar-triangles argument in Fig. 4:

$$\phi(z_v) = \frac{1}{q(z_v)} = \frac{1}{s} \cdot \frac{z_k}{z_v} = \rho \cdot \frac{z_k}{z_v}, \tag{7}$$

where $q(z_v)$ is the integration grid spacing at distance $z = z_v$ from the eyepoint and $1/s = \rho$.

The maximum volume raster frequency $\tilde{\omega}_{\max}(z_v)$ that the scaled filer h will pass can be derived from the scaling property of the Fourier transform [10]:

$$h\left(\frac{z_k}{z_v} y\right) \leftrightarrow \left|\frac{z_v}{z_k}\right| H\left(\frac{z_v}{z_k} \omega\right). \tag{8}$$

Therefore, we have

$$\tilde{\omega}_{\max}(z_v) = \omega_{\max} \cdot \frac{z_k}{z_v}. \tag{9}$$

Starting with (1): $\rho \geq 2\omega_{\max}$, it is now straightforward to conclude [11] that

$$\phi(z_v) \geq 2\tilde{\omega}_{\max}(z_v). \tag{10}$$

This derivation says that if the volume raster has sampled the function above the Nyquist limit, our antialiasing technique provides enough low-pass filtering so that sample-rate aliasing is not introduced when the volume raster is resampled onto the integration grid. Note that this derivation only deals with the prealiasing that results from an inadequate sampling rate — it does not address the aliasing or blurring effects which result from using a nonideal reconstruction filter.

3.5 Results From Applying Our Antialiasing Algorithm

In our implementation of this algorithm, we make use of rendering hardware to quickly draw the splats, in a manner similar to [1] and [16]. For each splat, we draw a polygon in world space centered at the voxel position. Depending on their z-coordinate, some polygons must be rotated so they are perpendicular to the ray passing from the eye point through the voxel position, while others can be left aligned with the x-y plane (see Section 4.2). The splat kernel is precomputed and stored in a 256 × 256 table which is texture mapped onto the polygon by the rendering hardware. We use the optimal cubic spline splat kernel reported in [1]. We attenuate the alpha channel of the polygon according to the value of $(z_k/z_v)^2$ and, then, composite the semitransparent splat polygon into the screen buffer.

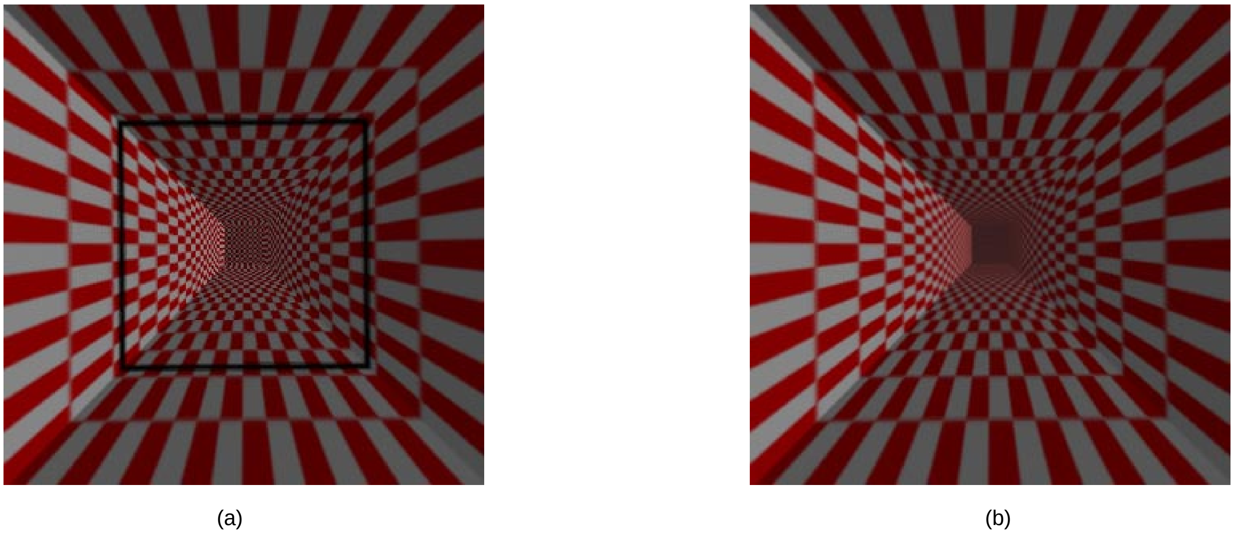


Fig. 5. Rendered image of a tunnel dataset with a checkerboard pattern. (a) Rendered with standard splatting; the black line is drawn at distance z_k. (b) Rendered with antialiased splatting.

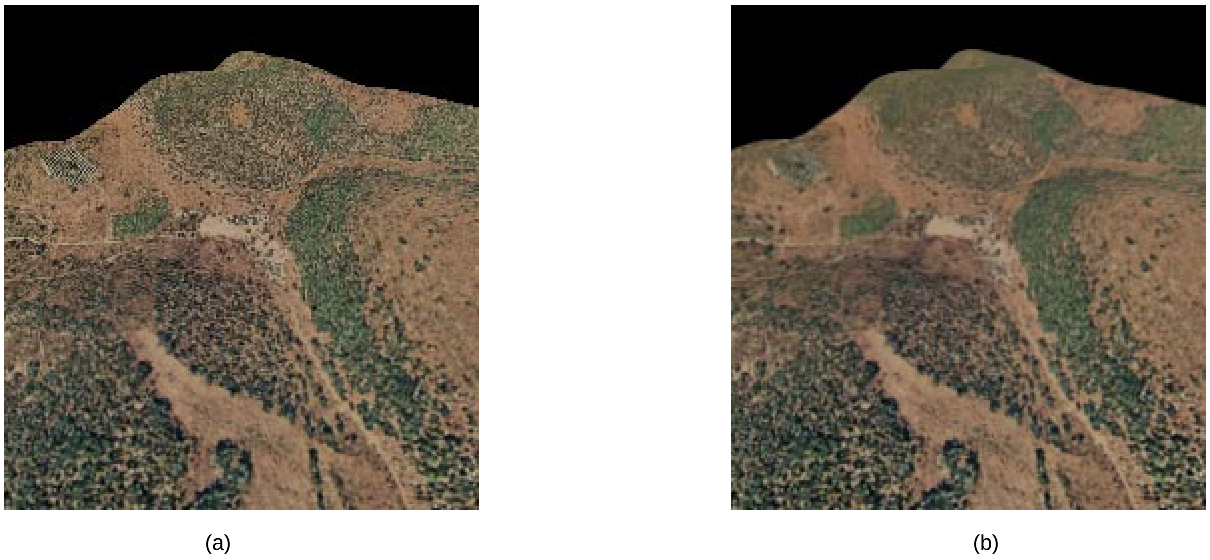


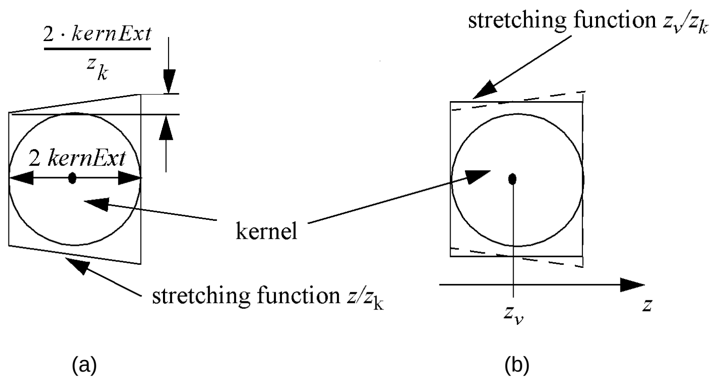
Fig. 6. Rendered image of a terrain dataset from satellite and mapping data. (a) Rendered with standard splatting. (b) Rendered with antialiased splatting.

Our renderer is a modified version of the “splat renderer” reported in [16]. For a given volume, we extract and store a subset of the voxels. For each voxel, we evaluate a transfer function $t = F(\nabla, \rho)$, where ∇ and ρ are the gradient and density of the voxel, respectively; we include the voxel in the subset if t exceeds a user-defined threshold. We store this volume subset as a 2D array of splat rows, where each row contains only the extracted voxels. Each row is implemented as an array of voxels, but the voxels are not necessarily contiguous, and, so, we must store each voxel’s location and normal vector. In general, each row may contain a different numbers of voxels. Despite not storing the empty voxels, we can still traverse this data structure in either a back-to-front or front-to-back order.

Fig. 5 shows a 160 × 160 × 300 volume consisting of a hollow tube, where alternate squares are colored either red or white to create a checkerboard effect. In Fig. 5a, a black line is drawn at the distance z_k; beyond this line, there is more than one voxel per pixel. As expected, Fig. 5a shows strong aliasing effects, but these are smoothed out in Fig. 5b. Fig. 6 shows a 512 × 512 × 103 volume containing a terrain dataset acquired from a satellite photograph and a corresponding height field. Each column of splats is given the color of the corresponding pixel from the satellite photograph. Fig. 6a shows strong aliasing in the upper half of the image (containing about 90 percent of the data), but, in Fig. 6b, these regions have been smoothed out.

4 Analysis of Perspective Splatting Errors

It is well known that the pre-integration of the 3D interpolation kernel into a 2D footprint is what achieves the high-efficiency gains of the splatting method. However, as is also well known, this use of pre-integrated footprint tables separates the reconstruction integration from the compositing operation. The 3D reconstruction kernel is composited as a



$\frac{2 \cdot kernExt}{z_k}$
2 kernExt
kernel
stretching function z/z_k
stretching function z_v/z_k
z_v
z
(a)
(b)

Fig. 7. (a) Stretching the splatting kernel according to the perspective stretching function z/z_k, which has the form of a trapezoid. (b) Stretching the kernel by the function z_v/z_k, which is a box.

whole, and not piecewise as a part of an interpolated sample along a viewing ray. This may lead to the image. To eliminate these errors, Westover [15] suggests the use of a series of compositing sheets parallel to the image plane. In each such sheet, the partial contributions of all intersecting kernels are added, and the sheets are then composited back-to-front. Similar to recent approaches that utilize 3D texture mapping hardware [12] for volume rendering, this method undoes many of the savings that result from the kernel's complete pre-integration. A compromise can be achieved by adding the entire 2D footprint into sheet buffers aligned parallel to a volume face, which are subsequently composited in back-to-front order [14]. It should be added that in the case of summed X-ray rendering, compositing does not take place and thus the accumulation of splats render a very accurate image (given a well integrated footprint [8]).

In this section, we will not attempt to characterize the errors that are due to the pre-integration problem just discussed. Rather, we shall investigate the errors that arise from the use of pre-integrated footprints in the context of perspective rendering. As we know, in contrast to orthographic rendering, in perspective rendering, the sight rays are no longer parallel, but diverging. Since there is, at least so far, no known efficient method that would allow the use of one generic footprint table for all different ray fan orientations that may traverse a kernel, we must resort to a footprint table whose ray integrals are due to integrations along a set of parallel trajectories. This necessarily will lead to errors.

There also seem to exist some uncertainties in the literature on how to position the splat footprint in eye space. Let us again think of the footprint as a polygon with the footprint function texture-mapped onto it, as was first implemented by Crawfis and Max [1]. Then the question is: How should this footprint polygon be positioned in eye space? In the previous sections, we have always aligned the footprint (splat) with the *x-y* plane. But, is this correct? It is unclear from Westover's description in [14] if he positioned all footprints the same way, coplanar to the sheet buffer planes, or if he individually aligned each voxel footprint perpendicular to the ray *eye – voxelCenter*. The former obviously saves a good deal of computations, since we don't have to compute a separate rotation matrix for each voxel.[1]

Westover [14] describes his splatting technique within the more general framework of volume grids with unequal scaling in the three grid directions. In these noncubic grids, ellipsoidal kernels and elliptical footprints are mapped to the image plane instead of spherical kernels and circular footprints in the cubic case. However, there exist viewpoints for which the mapping of these elliptical kernels is only approximate [15], a situation that is bound to be amplified when doing perspective mapping. We can avoid this problem by realizing that noncubic grids do not have to be rendered as such. As was mentioned by Mueller and Yagel [8], we can warp any grid with nonuniform scaling into a cubic one and, as long as the shading calculations are performed in the original grid, we can use spherical kernels for all grid projections. Hence, we will limit our discussion to cubic volume grids only.

As discussed in earlier sections, antialiased splatting in perspective is only possible by stretching the footprints by some amount proportional to their distance from the viewing plane. This leads to a distortion of the formerly spherical kernels in world space, which brings with it a whole new set of approximations necessary to maintain good efficiency. Recall that the kernel stretching is only required for voxels positioned at $z > z_k$.

In the following, we will discuss the errors and favorable approximations that come with footprint alignment for both volume regions, but first let us determine how the correct stretching function is computed and approximated.

4.1 Errors in Approximating the Kernel Stretching Function

We have shown earlier that, for voxels with $z > z_k$, the splatting kernel (or interpolation kernel, if we use raycasting) must be stretched above its normal size. To be specific, the stretching must occur according to the function z/z_k. This means that it is not correct just to stretch the 2D splat obtained by pre-integrating the standard 3D kernel function. Rather, we must perform the stretching along the full 3D interpolation kernel function according to a ramp function of slope $1/z_k$ (see Fig. 7a). Then, after the stretching has been

1. We should mention in this context that the need to compute a rotation matrix for each voxel only exists for the object-order splatting approaches. The ray-driven splatting technique described by Mueller and Yagel [8], on the other hand, does not have this requirement as it inherently orients the footprint polygons perpendicular to the sight ray.

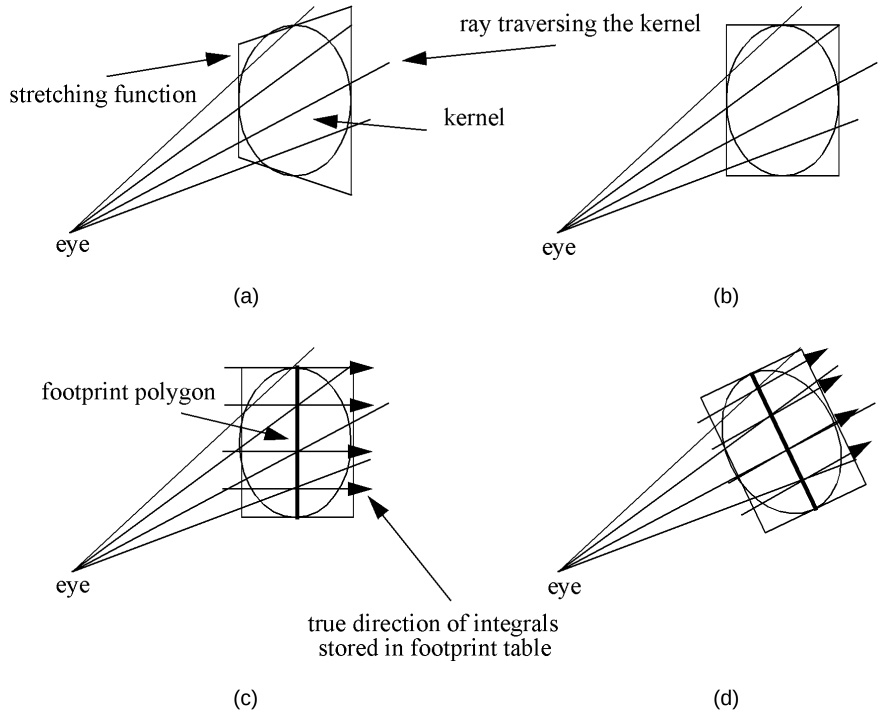

ray traversing the kernel
stretching function
kernel
eye
eye
(a)
(b)
footprint polygon
eye
eye
true direction of integrals stored in footprint table
(c)
(d)

Fig. 8. Different stretchings of a rotationally symmetric kernel and possible placements of its footprint polygon in world space (2D case). (a) Trapezoidal kernel stretching. (b) Approximating the trapezoid of (a) with a box, the result is an elliptical kernel that is not rotationally symmetric. (c) Aligning the footprint polygon (thick line) with the volume grid, the result is that the indexed ray integrals do not correspond to the true ray integrals. (d) Aligning the footprint polygon orthogonally to the ray *eye* – (y_v, z_v), which rectifies the problem of (c) in good approximation, however, also tilts the kernal stretching function.

performed, the kernel can be integrated into the footprint.

But, how is this done in practice? The relative kernel distortion varies depending on the kernel's z-coordinate. Simply pre-integrating the distorted kernel for one kernel center location $z = z_1$ and then scaling this footprint for another kernel at $z = z_2$, $z_2 \neq z_1$, based on the ratio z_2/z_1, does not yield the correct footprint of a distorted kernel at that location. If we wanted to use preintegrated footprints that bear the correct kernel distortion, we would have to use a different footprint for each kernel center z-location. This is obviously impractical. However, we realize that in most cases, $z_k \gg 2 \cdot kernExt$. For instance, for a 256^3 volume, a 256^2 image, and a 30° cone angle, the factor is never larger than 0.02 volume units. Thus, we may stretch the kernels by a box instead of a trapezoid (see Fig. 7b) without committing much of an error. This in turn enables us to use the generic footprint polygons, scaled up in the y- (and x-) direction depending on their location along z.

4.2 Errors in Orienting the Footprint in World Space

Fig. 8 shows different kernel orientations, ray traversals, and stretching functions for the 2D case. Fig. 8a shows the ideal situation: The splatting kernel is stretched along a ramp function with slope $1/z_k$ and the rays integrate the kernel along divergent paths. The perspective transform that takes this kernel $h(y, z)$ located at (y_v, z_v) from world space (y, z) into perspective space (y_p, $z_p = z$) is shown in Table 1, row 1. We saw in the previous section that the trapezoidal stretching function can be simplified with good approximation by a box function. This is shown in Fig. 8b, and the perspective transform for this kernel is listed in the second row of Table 1. However, this configuration still requires a new footprint preintegration for each kernel location, even if the traversing rays were parallel. This is because the integrals of the rays traversing such a stretched kernel are different for each (y_v, z_v) position, since the stretched kernels are no longer rotationally symmetric. We have two options that would enable us to use the same footprint (with different amounts of stretching) for all voxels:

1) Align each stretched footprint polygon with the y-axis of the grid (Fig. 8c and row 5 in Table 1), or
2) rotate each stretched footprint polygon so it is perpendicular to the ray *eye* – (y_v, z_v) (Fig. 8d and row 3 and 4 in Table 1).

Fig. 8c shows the situation for the grid-aligned footprint. We observe that the paths of the ray integrals retrieved from the table are rather different from the paths of the rays traversing the kernel. Thus, the indexed ray integrals will be very different from the true ray integrals. This error is largest for rays that maintain a large angle with the viewing axis; however, it is relatively small for voxels that are traversed by rays that are nearly parallel to the viewing axis. The latter is the case for voxels close to the viewing axis and for voxels far away from the eyepoint. We can rectify the problem illustrated in Fig. 8c by aligning the stretched footprint polygons with the ray *eye* – (y_v, z_v), as is shown in Fig. 8d. Now, the rays are traversing the kernel in roughly the same direction at which they were integrated. However, we are still left with the approximation that we cannot efficiently store ray integrals that are due to the spatially variant ray fan traversing the kernels. Instead, we have to store ray integrals that traverse the kernel along parallel lines. (Com-

TABLE 1
EQUATIONS THAT TRANSFORM A KERNEL LOCATED AT (y_z, z_v) FROM WORLD SPACE (y, z) INTO PERSPECTIVE SPACE $(y_p, z_p = z)$

row	integration, stretching footprint alignment	perspective kernel transform equation
1	correct integration, trapezoid stretch function	$\frac{z_k}{z} \cdot h\left(\frac{z_k}{z} \cdot \left(\frac{y_p z}{z_k} - y_v\right), z - z_v\right) = \frac{z_k}{z} \cdot h\left(\left(y_p - \frac{y_v z_k}{z}\right), z - z_v\right)$
2	correct integration, box stretch function	$\frac{z_k}{z_v} \cdot h\left(\frac{z_k}{z_v} \cdot \left(\frac{y_p z}{z_k} - y_v\right), z - z_v\right) = \frac{z_k}{z_v} \cdot h\left(\left(\frac{y_p z}{z_v} - \frac{y_v z_k}{z_v}\right), z - z_v\right)$
3	ray-perpendicular kernel, box stretch function, diverging ray traversal (ideal ray-perp. kernel)	$\frac{z_k}{z_v} \cdot h\left(\frac{z_k}{z_v} \cdot \left(\frac{-(z - z_v)x_v^2 + \left(\frac{y_p z}{z_k} - y_v\right)y_v^2}{\sqrt{x_v^2 + y_v^2}}\right), \frac{(z - z_v)y_v^2 + \left(\frac{y_p z}{z_k} - y_v\right)x_v^2}{\sqrt{x_v^2 + y_v^2}}\right)$
4	ray-perpendicular kernel, box stretch function, parallel ray traversal (practical ray-perp. kernel)	$\frac{z_k}{z_v} \cdot h\left(\frac{z_k}{z_v} \cdot \frac{(z_v y_p - y_v z_k)}{(z_v z_k - y_p y_v)}\sqrt{x_v^2 + y_v^2}, z - z_v\right)$
5	grid-aligned kernel, box stretch function, parallel ray traversal	$\frac{z_k}{z_v} \cdot h\left(\frac{z_k}{z_v} \cdot \left(\frac{y_p z_v}{z_k} - y_v\right), z - z_v\right) = \frac{z_k}{z_v} \cdot h\left(\left(y_p - \frac{y_v z_k}{z_v}, z - z_v\right)\right)$

pare also Table 1, row 3, which is the perspective transform for the ray-perpendicular kernel with a correct diverging ray traversal, which we will call the ideal ray-perpendicular kernel, and Table 1, row 4, which is the perspective transform of the ray-perpendicular kernel with the practical, parallel ray traversal.) Note, however, that rotating the kernel also rotates the stretching function away from its alignment with the viewing axis, which is clearly undesirable. In this respect, the alignment of Fig. 8c is a better one.

An interesting observation can be made when looking at the equations in rows 1 and 5 in Table 1. It turns out that aligning the footprint polygon with the y-axis is equivalent to replacing z with z_v in both denominators in the first form of the equation for the correct kernel integration with the trapezoidal stretching function. This turns this equation into the description of a circle centered at $(y_v \cdot z_k/z_v, z_v)$. Also notice that the equation of the perspective transform for a voxel kernel located in front of the z_k-plane is simply given by removing the scaling factor z_k/z or z_k/z_v in front of the equation and in the y-dimension of $h(y, z)$.

Fig. 9 shows a few perspective kernel transformations for the correct case depicted in Fig. 8a and the approximate cases of Figs. 8c and 8d. We used a Gaussian kernel with a standard deviation of 1.38 and truncated at a radius of 2.0. The volume was 512^3 and the cone half-angle $\varphi_c = 30°$. The top row of Fig. 9 shows the correctly distorted isocontour of the Gaussian kernel in perspective space. The second and third row depict the resulting perspective distortion of the Gaussian kernel if the two approximate footprint alignment methods are used. From the contour plots in row 2, we observe that the grid-aligned kernel function indeed maps to a circle in perspective space.

Finally, the bottom row shows the errors of the line integrals computed from the perspectively transformed kernel functions. Here, we can see what impact the approximate transformed kernel contours have on the accuracy of the splatted footprint function. The ray integral errors have been computed by subtracting at each footprint location the approximate ray integral value from the correct ray integral value and normalizing this value by the overall maximum value of the correct ray integral function. Since hardware assisted rendering requires the footprint function to be stored in an 8- to 16-bit texture map, this normalization helps us to get an idea about how many bits of error we have to expect for each of the approximations.

The isocontours of the perspectively distorted kernels of Fig. 9 suggest that, for voxels close to the z_k-plane, the ray-perpendicular footprint alignment is a good approximation to the correct method. However, for voxels further away from the z_k plane, the y-axis aligned footprint traversal seems to be the better choice. For voxels close to the main viewing axis, both approximations seem to be adequate. The computed normalized ray integral errors confirm this observation. It is only for voxels both close to the z_k-plane and far from the main viewing axis that the error for the grid-aligned footprint significantly exceeds the error for the ray-perpendicular footprint. We also see that there is not much difference between the ray integrals for the ideal ray-perpendicular kernel and the practical one.

For an 8-bit texture map, the bit-error for voxels located along the viewing axis is less than 1 bit for either kernel orientation. Close to the z_k-plane and far from the main viewing axis, the bit-error for the ray-perpendicular kernel is less than 2 bits, while the error for the grid-aligned kernel is more than 25 bits. Finally, the bit-error for voxels far from the z_k-plane is less than 8 bits for either of the two kernel orientations.

The plots in Fig. 10 give a road map of the errors for the entire volume. In these plots, the z_k-plane is located where $z = 0$, which is about 1/5 into the volume. In Fig. 10a, we show the maximum error (not normalized) of the ray integral at each voxel position for the grid-aligned kernel. (Note that,

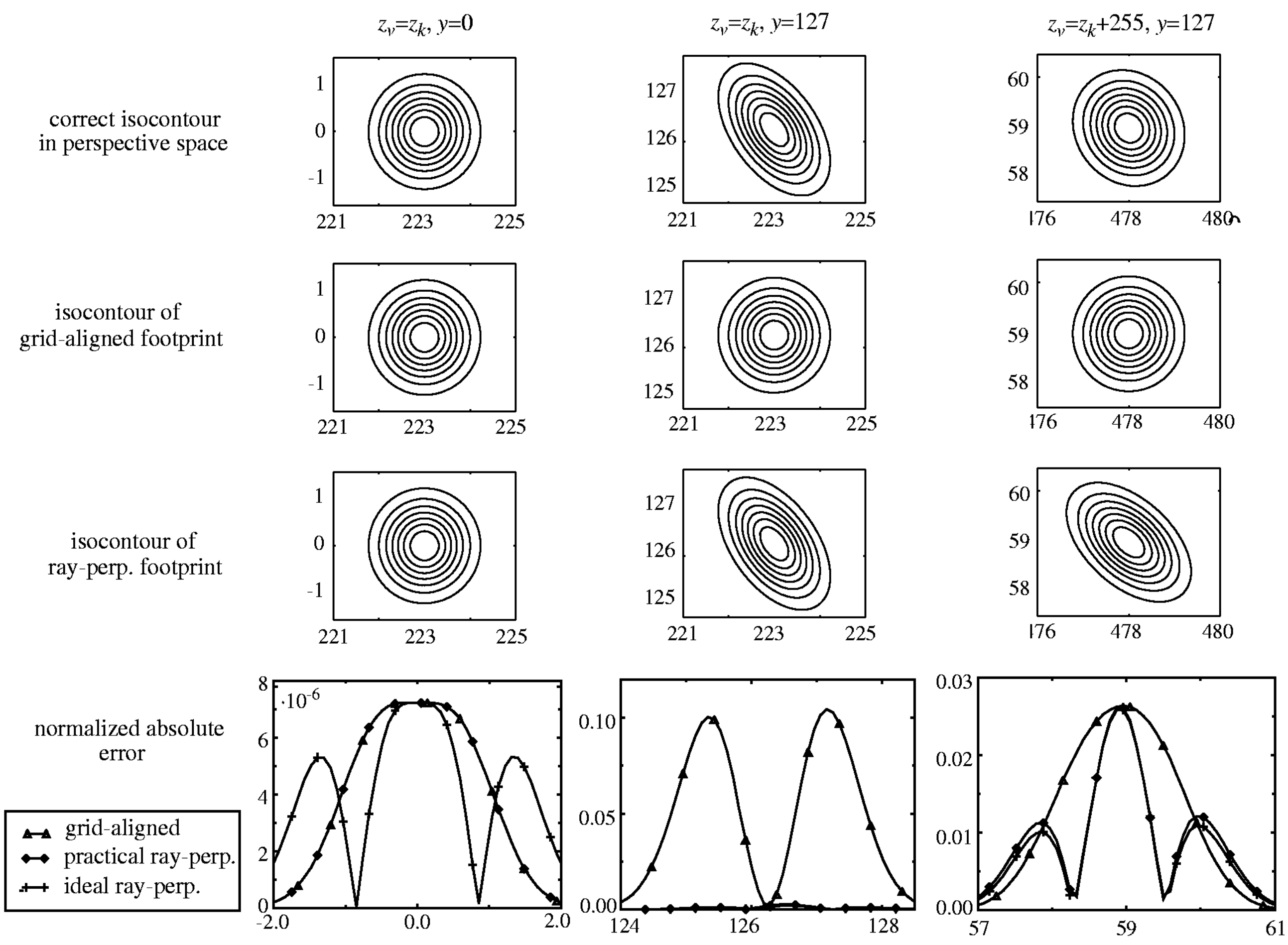


Fig. 9. The isocontours in the three top plots show the Gaussian kernel in perspective space. The bottom row shows how the errors of the ray integrals obtained by the approximate footprint methods of Fig. 8b (row 2) and Fig. 8c (row 3) compare with the correct ray integral of Fig. 8a (row 1). The absolute error is plotted, normalized by the maximum value of the ray integrals in the correct footprint. Column 1 is for voxels close to the eye and close to the viewing axis, column 2 is for voxels close to the eye, but far from the viewing axis, and column 3 is for voxel both far from the eye and far from the viewing axis.

for this and all other plots, a similar error surface is obtained for the cumulative integral error.) The isocontour lines on the bottom plane of the plot indicate the projected trajectories of equal error. We see that the maximum error for the grid-aligned kernel decays quickly for $z > z_k$, but is very large before that. Now, consider Fig. 10b, where we plot the maximum error for the ray-perpendicular kernel (the one that stores parallel ray integrals). Here, we see that the error is very small for $z < z_k$, but immediately gets larger for $z > z_k$ at the volume boundaries. Note, however, that these errors are still less than 5 percent of those of the grid-aligned kernel. The remaining error is due to the out-of-plane rotation of the kernel stretching function. The error decays as we go further back into the volume, since in these regions the kernel's stretching box is rotated less. On the other hand, the error for the grid-aligned kernel decays for larger z since the traversing rays become more parallel to each other and the ray slopes become smaller.

Fig. 10c gives the difference of the maximum errors of the two practical alignment methods, ray-perpendicular and grid-aligned. The isocontour lines indicate what error difference we accept if we switch from the computationally more expensive ray-perpendicular kernel to the grid-aligned kernel. The almost straight isocontour farthest back indicates when the grid-aligned kernel footprint is actually slightly better than the ray-perpendicular kernel footprint (at least for the maximum integral error). The same isocontour suggests that there is a $z_s > z_k$ after which we can safely switch to grid-aligned kernels, minimizing the committed error. For a cone half-angle of 30° and a 512^3 volume grid with a 512^2 image, we found this $z_s \approx z_k + 70$.

Finally, Fig. 10d shows the maximum integral errors when we approximate the trapezoidal stretching function with a box, but still use the correct ray traversal of Fig. 8b. The errors are very small everywhere, and the plot confirms our earlier intuition that the box approximation is a good one.

5 A Temporal Antialiasing Technique for Splatting

5.1 Motion-Blurred Splats

For dynamic scenes or fast interactive movements, severe aliasing can again come into play due to improper sampling of the time domain. Here again, we need to reduce the frequency

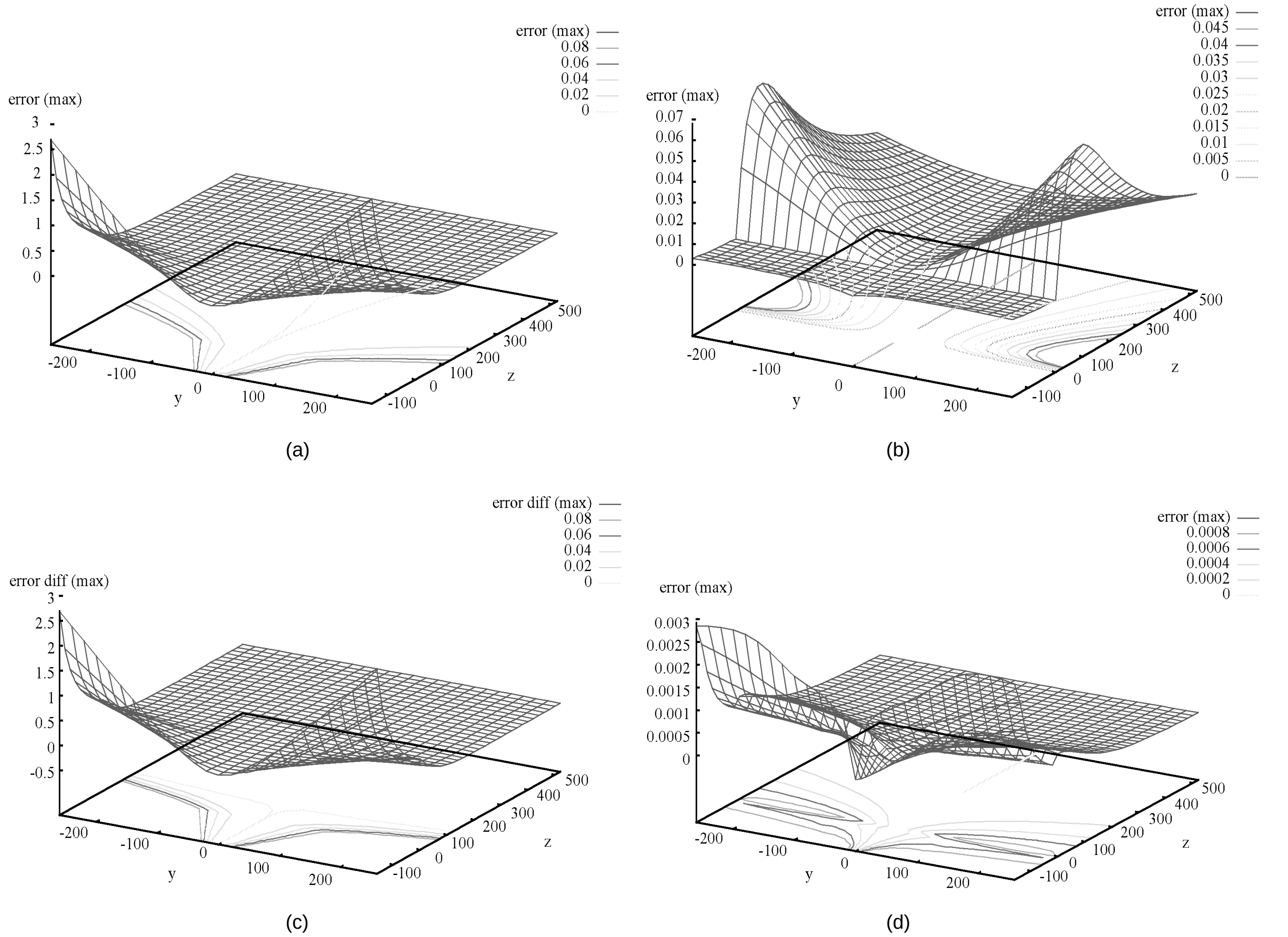


Fig. 10. Plots: (a) Maximum error for the grid-aligned kernel. (b) Maximum error for the practical ray-perpendicular kernel. (c) Difference of the maximum errors of the two kernel alignments (the iso-contours on the bottom planes denote lines of equal error). (d) Maximum error for the ideal kernel ray traversal when the trapezoidal stretching function is approximated by a box.

of the volume where this is a problem. This process is commonly called motion blur. It has been studied in the computer graphics literature for animations, and can be considered as the integration of the reflectance and visibility over the length of time a virtual camera shutter remains open. The fundamental problem that most techniques have difficulty solving is the changing hidden surfaces and visibility. For volume rendering, we also have this problem with semitransparent or opaque compositing. However, for X-ray-like or emission volume models, Max [7] and Crawfis et al. [2] have shown that the compositing order is immaterial. Here, we present some initial work on extending splatting to handle motion blur. Our algorithm will not address the visibility problem, but simply address the issue of the integrated energy across the time domain. The issue of motion blurring in volume rendering has not yet been addressed and presents a very fruitful area of future research.

Our algorithm approximates the motion of a volume defined on a raster grid as a linear motion of the reconstruction basis functions at each voxel. Fig. 11 illustrates this process and the resulting footprint silhouette. For each voxel, we associate a linear motion vector. This vector is transformed to eye space and a projected vector direction and length is determined on the viewing plane. For a radi-

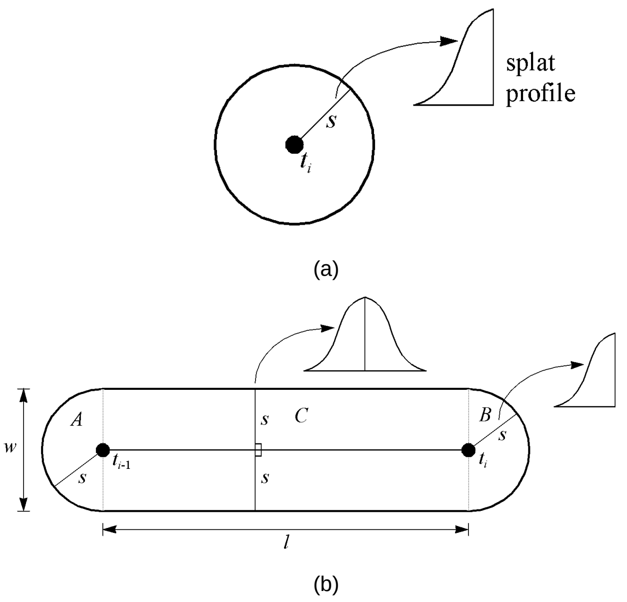


Fig. 11. The construction of a nonblurred and a motion-blurred splat. (a) A nonblurred splat (splat 1). (b) A motion-blurred splat (splat 2).

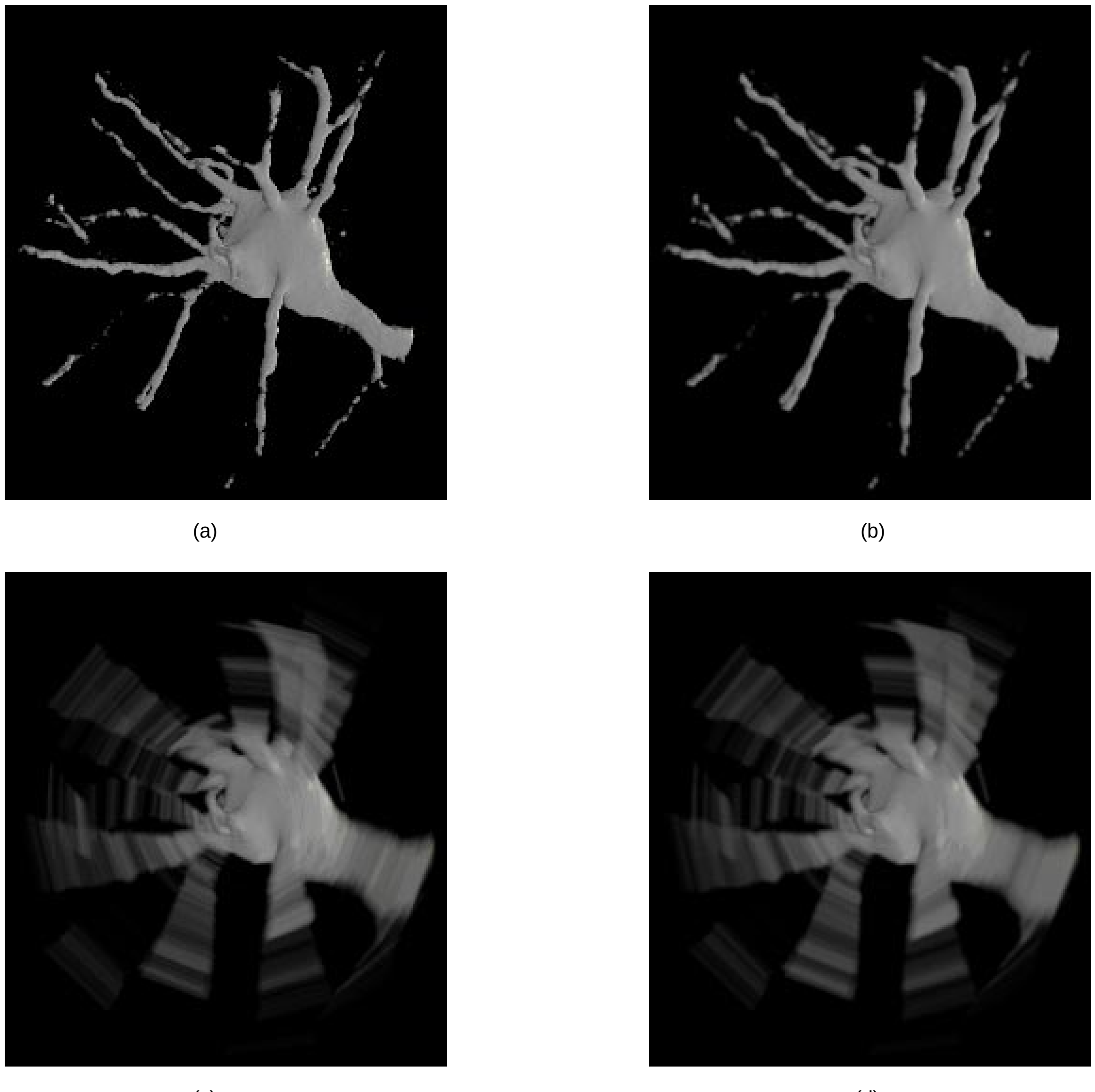


Fig. 12. Rendered image of a nerve dataset acquired from confocal microscopy. (a) Rendered with standard splatting. (b) Rendered with antialiased splatting. (c) Rendered with motion-blurred splats. (d) Rendered with antialiased, motion-blurred splats.

ally symmetric footprint function, we further approximate this process by smearing (or stretching) the footprint function in the direction of the projected velocity vector.

Fig. 11 shows how the motion blurred splats are drawn. Fig. 11a shows a nonblurred splat (labeled splat 1), while Fig. 11b shows a motion-blurred splat (labeled splat 2). Splat 1 is drawn at position t_i. The splat is radially symmetric, so its footprint is a circle and radial profile s follows the splat's reconstruction kernel. Splat 2 is drawn between positions t_{i-1} and t_i. The original circular footprint is cut in half and stretched between t_{i-1} and t_i. This results in the two half-circles A and B, connected by the rectangle C with width w and length l. A and B have the same radial profile s as splat 1. C has the profile s along any line perpendicular to the motion vector $\overline{t_{i-1}t_i}$.

For this process to work, we need the amount of energy that a splat contributes to remain constant under high velocities. In other words, the integrated intensity of a motion-blurred splat should be the same as the integrated energy of a single stationary splat. For our current, rather simple, scheme, we adjust the energy of the motion-blurred splat as a function of the increased area the footprint's extent will occupy. If the area of our static footprint extent is A_1, then the area of the motion-blurred splat can be expressed as $A_2 = A_1 + 2ls$, where l is the distance between t_{i-1} and t_i and s is the radial splat radius. We choose to adjust the splat opacity for this process. This easily translates into a scaling of the opacity as:

$$\alpha_2 = \frac{A_1}{A_1 + 2ls}\alpha_1. \quad (11)$$

For arbitrary projected velocity vector directions, a similar scheme to that of Crawfis and Max [1] is employed to rotate the motion-blurred splat in the proper direction.

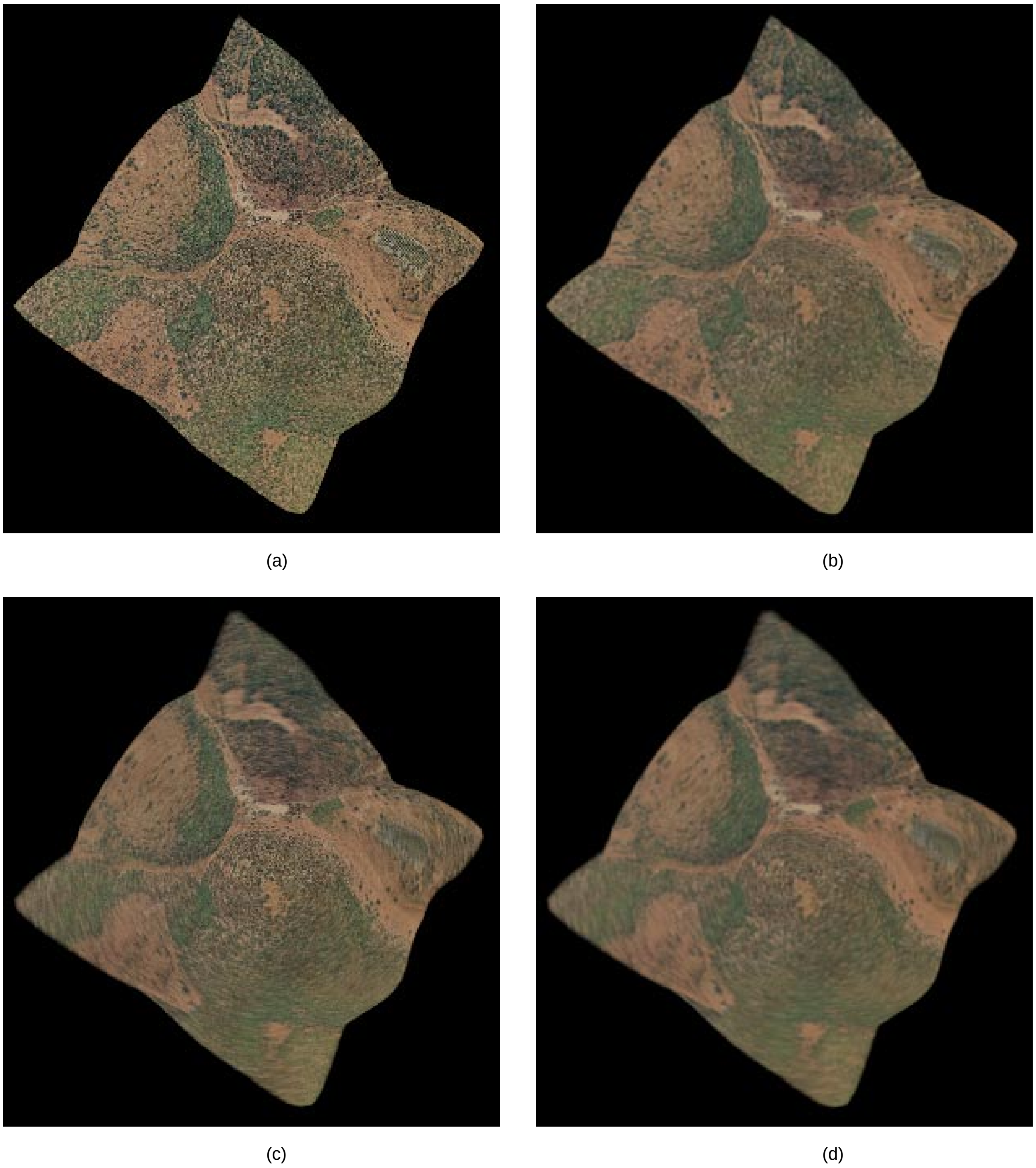


Fig. 13. Rendered image of a terrain dataset from satellite and mapping data. (a) Rendered with standard splatting. (b) Rendered with antialiased splatting. (c) Rendered with motion-blurred splats. (d) Rendered with antialiased and motion-blurred splats.

5.2 Results From Applying Our Antialiasing and Motion-Blur Techniques

Fig. 12 shows an example of both the antialiasing and motion-blur techniques applied to an animation of a nerve cell spinning rapidly. This dataset is a 512 × 512 × 76 volume acquired from confocal microscopy. In Fig. 12a, the nerve is rendered with standard splatting—note the aliasing artifacts along the arms of the cell. Fig. 12b shows the same cell rendered with antialiased splatting; the aliasing artifacts are now smoothed out. Fig. 12c shows the cell rendered with motion-blurred splats, while Fig. 12d shows the cell rendered with both antialiased and motion-blurred splats. For an object with this much movement, the motion blur dominates, and there is not much of a visible difference between Figs. 12c and 12d.

Fig. 13 shows frames from an animation of the same terrain dataset as Fig. 6, spinning about its center, rendered with standard splats, antialiased splats, motion-blurred splats, and both antialiased and motion-blurred splats. When rendered from this perspective, all of the voxels map to subpixel sizes. Because the terrain image contains high spatial frequencies, the animation of the spinning dataset shows considerable

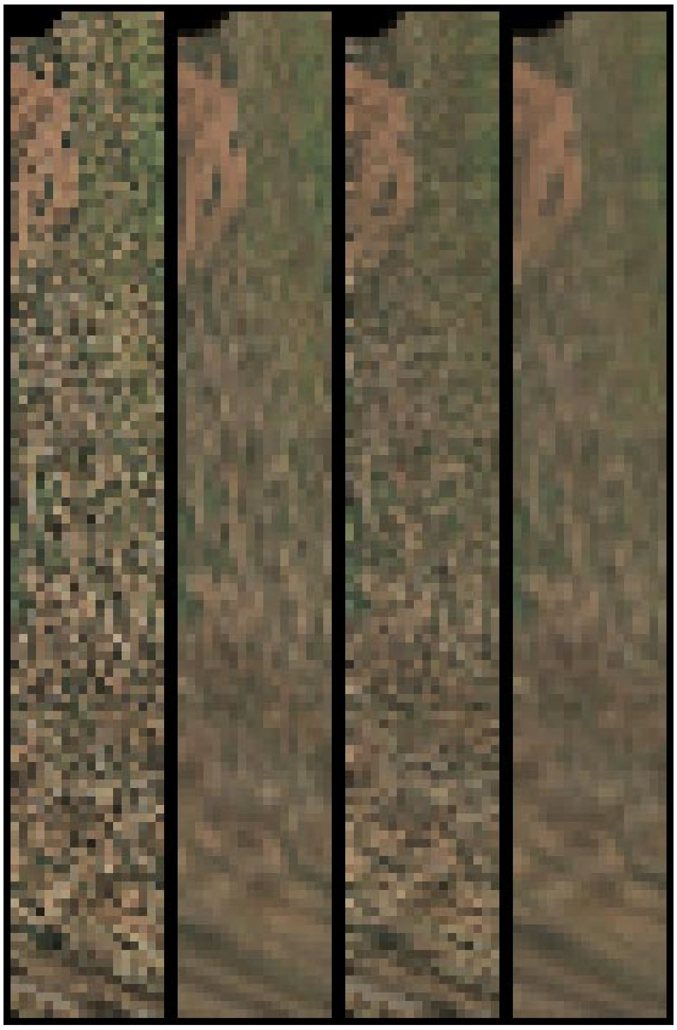


Fig. 14. Animated sequence of the terrain dataset in Fig. 13. (a) Sequence from aliased animation. (b) Sequence from antialiased animation. (c) Sequence from motion-blurred animation. (d) Sequence from antialiased, motion-blurred animation.

flickering and scintillation artifacts in Figs. 13a and 13c. In addition, Figs. 13a and 13b are rendered with the equivalent of a camera with an instantaneous shutter and, so, suffer strobing artifacts. Both of these temporal artifacts are greatly reduced in the animation of Fig. 13d.

Fig. 14 attempts to display these temporal properties in a static image. It shows how a single column from the terrain animation varies over time. To create Fig. 14, a rectangular region one pixel wide extending from the center to the upper edge of each image was taken from successive frames of the animation. These columns were abutted and, then, expanded into 5 × 5 blocks to form Fig. 14. Thus, the rows of Fig. 14 are a spatial representation of the temporal progression of the animation. Note the almost random pixel colors that occur across the rows of Fig. 14a—these represent the strong flickering and scintillation artifacts visible in this animation. As expected, the same random pixel colors occur in Fig. 14c, but are less pronounced toward the top of the column. This reflects the subjective appearance of the animation from Fig. 13c, where the terrain color is relatively static around the outer edges but suffers considerable flickering and scintillation in the center. In comparison, the rows of Figs. 14b and 14d show smooth variations of color, and indicate the absence of these flickering and scintillation artifacts. Figs. 14c and 14d show more blurring toward the top of the column compared to Figs. 14a and 14b, which indicate the faster motion of the outer edge relative to the center. Subjectively, the strobing artifacts are greatly reduced in the animations from Figs. 13c and 13d.

6 Future Work

Currently, our implementations for this work focus on the flexibility of splat orientation and placement. As such, our implementations are far from optimal in their performance; this is one of the main reasons for the lack of performance statistics presented here. Future work is ongoing to find efficient and optimal implementations of the splatting process as a whole. The antialiased splats offer several possibilities for very efficient implementations. With a fixed screen extent, a very fast pixel *bitblt* operation may be possible. Of course, the splats are not necessarily centered at a pixel, and we are conducting research to rectify these deficiencies. Furthermore, we have determined the amount of blurring for the antialiasing under the assumptions that an ideal reconstruction function is used and the sampled data exhibits frequencies near the maximum dictated by the volume grid. Future work will examine the amount of blurring based on the reconstruction kernel chosen.

The new perspective error analysis has led us on a path to determine regions where different splatting operations and algorithms should be utilized to minimize the resulting visual artifacts. While the perspectively correct integration may be very complex, if the region where this is required is small enough, then, perhaps, a precomputed table of perspective splats can be built, quantizing the necessary error.

There is a wealth of research potential in the motion-blurred splats. We are currently examining the integration of the motion-blurred splats in a more analytical manner. In addition, it is well known that splatting suffers from visibility problems which lead to a popping artifact as the sorting of the voxels changes. Future research will examine methods for efficiently handling these visibility problems, both alone and in the context of motion blur.

Acknowledgments

Special thanks go to Nelson Max for many helpful comments and last-minute clarifications. We acknowledge Daniel Cohen-Or for the terrain dataset and Noran Instruments for the confocal dataset, The Ohio Visualization Lab at The Ohio Supercomputer Center for computer access, and David Bressan and Rob Rosenberg of the CCS Scientific Visualization Lab at the Naval Research Laboratory for assistance with preparing the images, the animations, and the paper.

REFERENCES

[1] R. Crawfis and N. Max, "Texture Splats for 3D Scalar and Vector Field Visualization," *Proc. Visualization '93*, pp. 261-266, San Jose, Calif., 25-29 Oct. 1993.

[2] R. Crawfis, N. Max, and B. Becker, "Vector Field Visualization," *IEEE Computer Graphics and Applications*, vol. 14, no. 5, pp. 50-56, Sept. 1994.

[3] R.A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering," *Computer Graphics (Proc. SIGGRAPH)*, vol. 22, no. 4, pp. 65-74, Aug. 1988.

[4] P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation," *Computer Graphics (Proc. SIGGRAPH)*, pp. 451-458, July 1994.

[5] D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics (Proc. SIGGRAPH)*, vol. 25, no. 4, pp. 285-288, July 1991.

[6] X. Mao, "Splatting of Non Rectilinear Volumes Through Stochastic Resampling," *IEEE Trans. Visualization and Computer Graphics*, vol. 2, no. 2, pp. 156-170, June 1996.

[7] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99-108, June 1995.

[8] K. Mueller and R. Yagel, "Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach," *Proc. Visualization '96*, pp. 65-72, San Francisco, 27 Oct.-1 Nov. 1 1996.

[9] K.L. Novins, F.X. Sillion, and D.P. Greenberg, "An Efficient Method for Volume Rendering Using Perspective Projection," *Computer Graphics (Proc. San Diego Workshop Volume Visualization)*, vol. 24, no. 5, pp. 95-102, Nov. 1990.

[10] A.V. Oppenheim and R.W. Shafer, *Digital Signal Processing*. Englewood cliffs, N.J.: Prentice Hall, 1975.

[11] J.E. Swan II, K. Mueller, T. Möller, N. Shareef, R. Crawfis, and R. Yagel, "An Antialiasing Technique for Splatting." *Proc. IEEE Visualization '97*, pp. 197-204, 1997.

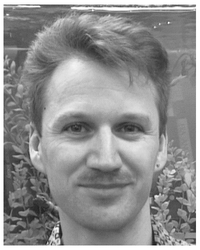
[12] A. Van Gelder and K. Kim, "Direct Volume Rendering with Shading via 3D Textures," *Proc. 1996 Symp. Volume Visualization*, pp. 23-30, San Francisco, 27 Oct.-1 Nov. 1996.

[13] L.A. Westover, "Interactive Volume Rendering," *Proc. Volume Visualization Workshop*, pp. 9-16, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, 18-19 May, 1989.

[14] L.A. Westover "Footprint Evaluation for Volume Rendering," *Computer Graphics (Proc. SIGGRAPH '90)*, vol. 24, no. 4, pp. 367-376, Aug. 1990.

[15] L.A. Westover, "SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm," PhD dissertation, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, 1991.

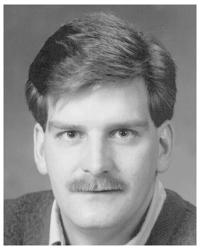
[16] R. Yagel, D.S. Ebert, J. Scott, and Y. Kurzion, "Grouping Volume Renderers for Enhanced Visualization in Computational Fluid Dynamics," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 2, pp. 117-132, June 1995.



Klaus Mueller received a BS degree in electrical engineering from the Polytechnic University of Ulm, Germany, in 1987, and an MS degree in biomedical engineering from The Ohio State University in 1990. Previous employment includes an internship at Mercedes Benz, Germany, and research and development engineer positions at Medical Devices, Minneapolis, and Bopp and Reuther, Mannheim, Germany. Mueller is currently working on a PhD degree in computer and information science at The Ohio State University, where he also holds a graduate research appointment funded by General Electric. His interests reside in the field of computer graphics, in particular, volume graphics as applied to the visualization of medical image data. His current focus is on the development of iterative algorithms for fast and accurate 3D and 4D reconstruction from limited sets of 2D cone-beam projections. The design of an efficient and effective PACS system is another area of his current academic activities. For more information, see: http://chopin.bme.ohio-state.edu/~klaus.

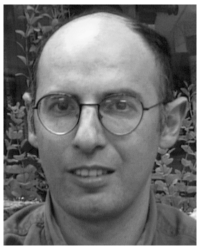
Torsten Möller received a Vordiplom (BSc degree) in mathematical computer science from the Humboldt University in Berlin, Germany, in 1992, and an MSc degree in computer and information science from The Ohio State University in 1993. Previous employment includes summer positions at Hewlett-Packard Research Labs and the Lawrence Livermore National Laboratories in the Scientific Visualization group and an internship at mental images in Berlin, Germany. He is currently working on a PhD degree in computer and information science at The Ohio State University, where he also holds a graduate research appointment. His research interests reside in the field of computer graphics, in particular in applying mathematical methods and ideas from signal processing to evaluate the rendering process. His current focus is on the development of algorithms for fast and accurate rendering of unstructured three dimensional data sets. For more information, see http://www.cgrg.ohio-state.edu/~tmoeller.

J. Edward Swan II received his BS degree in computer science from Auburn University in 1989, and his MS and PhD degrees in computer science from The Ohio State University in 1992 and 1997, respectively. He is a research scientist at the Naval Research Laboratory in Washington, D.C. He is currently conducting research in the area of battlefield visualization, where he is applying volume graphics techniques to render large terrain databases as well as studying usability issues of battlefield visualization systems. Previously, he was employed as a graduate research and teaching assistant at The Ohio State University, where he conducted research in computer graphics, volume graphics, virtual reality, and human-computer interaction. For more information see http://www.ait.nrl.navy.mil/people/swan/.



Roger Crawfis received his BS in computer science and applied mathematics from Purdue University in 1984, and his MS and PhD degrees in computer science from the University of California, Davis, in 1989 and 1995, respectively. He is an assistant professor at The Ohio State University. His research interests include scientific visualization, computer graphics, and volume rendering. Prior to joining OSU, Dr. Crawfis was the Graphics Technology Group Leader at the Lawrence Livermore National Laboratory, where he was in charge of coordinating several visualization projects for the past 12 years. He has published many research papers on scientific visualization and the volume rendering of scalar and vector fields.

Naeem Shareef is a graduate student pursuing a PhD in computer and information science at The Ohio State University. He received a BS in applied mathematics and computer science from Carnegie Mellon University in 1990, and his MS in computer and information science from The Ohio State University in 1992. His research interests include volume graphics, scientific visualization, image segmentation using neural networks, visualization of large datasets, medical applications for volumes, and voxelization. He has held a graduate research appointment at The Ohio Supercomputer Center since 1993 and currently holds an appointment there funded by Lockheed Martin. Previously, funding was provided by The Ohio State University Cancer Hospital Research Institute. His publications include work in voxelization using CSG, image and volume segmentation using a neural network, and volume graphics in the medical image visualization area. Currently, he is working on research in the area of visualization of large volume datasets. For more information see: http://www.cis.ohio-state.edu/~shareef.



Roni Yagel received his PhD in 1991 from the State University of New York at Stony Brook, where he was also a researcher in the Department of Anatomy and the Department of Physiology and Biophysics. He received his BSc cum laude and MSc cum laude from the Department of Mathematics and Computer Science at Ben Gurion University of the Negev, Israel, in 1986 and 1987, respectively. Dr. Yagel is an associate professor in the Department of Computer and Information Science and an adjunct assistant professor in the Advanced Computing Center for the Arts and Design and the Biomedical Engineering Center at The Ohio State University. He heads the Volume Graphics Research Group, which pursues research in computer graphics, volume graphics, scientific visualization, virtual reality, and image processing. His research and technical publications deal with various topics in volume graphics and visualization. His research interests also include algorithms for graphics, imaging, and animation, error control in image generation, and visualization tools for industrial and scientific applications. For more information see: http://www.cis.ohio-state.edu/~yagel.