Perspective Splatting

J. Edward Swan II, Klaus Mueller, Torsten Möller, Naeem Shareef, Roger Crawfis, and Roni Yagel

Abstract

Splatting is a popular direct volume rendering algorithm that was originally conceived and implemented to render orthographic projections. This paper describes an anti-aliasing extension to the basic splatting algorithm, as well as an error analysis, that make it practical to use for perspective projections. To date, splatting has not correctly rendered cases where the volume sampling rate is higher than the image sampling rate (e.g. more than one voxel maps into a pixel). This situation arises with perspective projections of volumes, as well as with orthographic projections of high-resolution volumes. The result is potentially severe spatial and temporal aliasing artifacts. Some volume ray-casting algorithms avoid these artifacts by employing reconstruction kernels which vary in width as the rays diverge. Unlike ray-casting algorithms, existing splatting algorithms do not have an equivalent mechanism for avoiding these artifacts. In this paper we propose such a mechanism, which delivers high-quality splatted images and has the potential for a very efficient hardware implementation. In addition, we analyze two numerical errors that arise with splatted perspective projections of volumes, and describe the rendering-time versus image-quality tradeoffs of addressing these errors.

Keywords and Phrases: volume rendering, splatting, direct volume rendering, resampling, reconstruction, anti-aliasing, perspective projection.

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, and splatting is one of several popular direct volume rendering techniques. The majority of images produced through direct volume rendering have used orthographic projections, in part because such projections are useful in many of the application areas (such as biomedical and fluid flow visualization) which have initially motivated work in volume rendering. Perspective projections offer a viewpoint which more naturally correlates to the way we perceive the physical world, and perspective projections are essential when it is desirable to “fly through” the data — flight simulators are one example. A perspective projection of a volume dataset gives another perceptual cue which can be employed when comprehending spatial relationships.

Any volume rendering algorithm which supports perspective projections has to deal with the problem of non-uniform sampling produced by diverging viewing rays. If not addressed this can result in potentially severe aliasing artifacts. Although other volume rendering approaches have dealt with this problem (e.g. ray-casting [21,23] and shear-warp [7,11]), to date the problem has not been addressed in the context of splatting. In this paper, we present a modification to the splatting algorithm which prevents the aliasing that arises from this non-uniform sampling. The same type of resampling problems 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 modified splatting algorithm also avoids aliasing in this situation.

We recently presented our algorithm at the *Visualization '97* conference [26]. The current paper is an expanded version of this conference publication. In addition to an extended explanation of the anti-aliasing technique itself (including a detailed description of the intuitive basis of the technique based on frequency and spatial domain diagrams), the current paper analyzes two errors that arise with splatted perspective projections of volumes, and describes the rendering-time versus image-quality tradeoffs of addressing these errors. It also discusses applications of the technique to traditional texture mapping, and other areas of future work.

In the next section we describe the splatting algorithm and related previous work, and then we give some advantages and disadvantages of splatting as compared to other volume rendering techniques. In Section 4 we describe our anti-aliasing technique and argue for its correctness. We follow this with implementation details and example images. In Section 6 we analyze perspective splatting errors, and in Section 7 we discuss our findings and indicate areas of future work.

2. Previous Work

The splatting technique has been used to directly render volumes of various grid structures [16,29] and for both scalar [13,29,30,31] and vector fields [5]. The basic algorithm, first described by Westover [29], projects each voxel to the screen and composites it into an accumulating image (Figure 1a). 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. Splatting is an *object-order* algorithm: the resulting image is built up voxel-by-voxel. This is in contrast to volume rendering by *ray-casting*, which is an image-order algorithm that builds up the resulting image pixel by pixel.

As each voxel is projected onto the image plane, the voxel's energy is spread over the image raster using a reconstruction kernel centered at the voxel's projection point (Figure 1b). This reconstruction kernel is called a "splat"; its name comes from the colorful analogy of throwing a snowball against a wall, with the spreading energy analogous to the “splatting” snow. Conceptually, the splat is considered a spherically symmetric 3D reconstruction kernel centered at a voxel. But because the splat is reconstructed into a 2D image raster, it can be implemented as a 2D reconstruction kernel. This 2D kernel, called a “footprint function”, contains the integration of the 3D kernel along one dimension (Figure 1c). Because the 3D kernel is spherically symmetric, it does not matter along which axis this integration is performed. The integration is usually pre-computed, and the footprint function is represented as a finely sampled 2D lookup table. The 2D table is centered at the projection point and sampled by the pixels which lie within its extent. Each pixel composites the value it already contains with the new value from the footprint table. Under certain conditions (regular volume grid spacing, orthographic view projection, radially symmetric splat kernel) the footprint table can be computed once and used unmodified for all voxels. Under different conditions, the footprint function will vary, and consequently must be re-computed for each view (when there is a non-symmetric kernel) and possibly for each voxel (when there is a perspective projection).

Recent work has extended the original splatting algorithm to achieve higher quality as well as faster rendering. To improve image quality, in later work Westover [30,31] first accumulates splats onto a 2D sheet that is 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. Laur and Hanrahan [13] change the size of a splat based upon the cell it represents in an octree representation of the volume. Mao [16] uses spherical and ellipsoidal kernels with varying sizes to splat non-rectilinear grids. Mueller and Yagel [19] 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 kernel types can generate higher quality images. Max [17] and Crawfis and Max [5] propose quadratic spline functions, optimized for certain conditions, as splat kernels.

To improve rendering speed, Westover [30] 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 [13] approximate splats with a triangle mesh and use graphics hardware to quickly scan convert the footprint. Crawfis and Max [5] and Yagel et al. [35] also use texture mapping hardware to quickly render splats represented as textures mapped to polygons. Splatting can also be accelerated by preprocessing the volume and culling voxels which will not contribute to the final image. Laur and Hanrahan [13] cull with an octree structure, and Yagel et al. [35] extract and store only the most visually significant voxels.

3. Advantages and Disadvantages of Splatting

In this section we compare splatting to other rendering algorithms. When listing the disadvantages of splatting, we distinguish between inherent problems and those that are due to inaccuracies in current splatting implementations.

3.1 Advantages of Splatting

The main advantage of splatting over ray-casting is that splatting is inherently faster. In ray-casting, reconstruction is performed for each sample point along the ray. At each sample point a k^3 convolution filter is applied, where k is the dimension of the filter. Even if, on the average, each of the n^3 voxels are sampled only once, ray-casting has a complexity of *at least* k^3n^3. In splatting, on the other hand, the convolution is precomputed, and every voxel is splatted exactly once. Each splat requires k^2 compositing operations. Therefore, one can expect a complexity of *at most* k^2n^3. This gives splatting an inherent speed advantage. An additional benefit is that one can afford to employ larger reconstruction kernels and improve the accuracy of splatting, incurring an $O(k^2)$ penalty instead of an $O(k^3)$ penalty.

Because splatting is an object-order rendering algorithm, it has a simple, static parallel decomposition [14,31], where the volume raster is evenly divided among the processors. It is more difficult to distribute the data with ray-driven approaches, because each ray might need to access many different parts of the volume raster.

Splatting is trivially accelerated by ignoring empty voxels. It can be accelerated further by extracting and storing just those voxels which contribute to the final image [35], which prevents traversing the entire volume raster. This is equivalent to similar acceleration techniques for volume ray-casting, such as space-leaping [34] or fitted extents [25], which accelerate ray-casting by quickly traversing empty space.

Because splatting generates images in a strict front-to-back or back-to-front order, observing the partially created images can give insight into the data which is not available from image-order techniques. In particular, with a back-to-front ordering, partial images reveal interior structures, while with a front-to-back ordering it is possible to terminate the rendering early [19].

Finally, splatting is the preferred volume rendering technique when the desired result is an X-ray projection image instead of the usual composited image [19]. This is because the summation of pre-integrated reconstruction kernels is both faster and more accurate than ray-casting approaches, which require the summation of many discrete sums. Creating X-ray projection images from volumes is an important step in the reconstruction algorithms employed by tomographic medical imaging devices [19,3], such as CT and PET. Indeed, several authors of the current paper have applied the splatting-based anti-aliasing technique reported here to the related problem of reconstructing 3D volumetric objects from a set of cone-beam projection data [20], with good results.

3.2 Inherent Disadvantages of Splatting

There are two disadvantages inherent to the splatting method. One is that while an ideal volume renderer first performs the process of reconstruction and then the process of integration (or composition) for the entire volume, splatting forces both reconstruction and integration to be performed on a per-splat basis [27,30] (see Figure 2). The result is incorrect where the splats overlap, and the splats must overlap to ensure a smooth image. This problem is particularly noticeable when the traversal order of the volume raster changes during an animation [30].

Another disadvantage of splatting lies in the ordering of the classification, shading, and reconstruction steps. For efficiency reasons, in splatting both (transfer function-based) classification and shading are usually applied to the data *prior* to reconstruction. This is also commonly done in ray-casting [12]. However, this produces correct results only if both classification and shading are linear operators. The result of employing a non-linear classification or illumination model may cause the appearance of pseudo-features that do not exist in the original data. For example, if we want to find the color and opacity at the center point between the two data values a and b using linear interpolation, we would compute $(C(a) + C(b))/2$ performing classification first, but the correct value would be $C((a + b)/2)$ performing interpolation first. Clearly, if C is a non-linear operator, these two results will be different. Requiring C to be linear generally means that the shading model can only model diffuse illumination. While methods exist for ray-casting that perform classification and shading *after* reconstruction [2,18,23], this is not possible in splatting.

3.3 Implementation-Based Disadvantages of Splatting

With a ray-casting volume rendering algorithm it is easy to terminate the rays early when using a front-to-back compositing scheme, which can substantially accelerate rendering. Although not reported in the literature, early termination could potentially be implemented for splatting by employing the dynamic screen mechanism [24] (also used by [11] for shear-warp volume rendering). Also, the ray-driven splatting implementation [19] can support early ray termination.

While ray-casting of volumes was originally implemented for both orthographic and perspective viewing, splatting was fully implemented only for orthographic viewing. Although for perspective viewing ray-casting has to include some mechanism to deal with the non-uniform reconstruction that is necessary with diverging viewing rays, for perspective viewing splatting needs to address several more inaccuracies. As is described in [19], when mapping the footprint polygon onto the screen an accurate perspective splatting implementation must:

(1) align the footprint polygon perpendicularly with the sight ray that goes through the polygon center, and

(2) ensure that the sight ray for every pixel that falls within the screen extent traverses the polygon at a perpendicular angle.

Both conditions are violated in the original splatting algorithm by Westover [29]. Mueller and Yagel [19] describe a voxel-driven splatting method that addresses condition (1), and a ray-driven splatting method that addresses both conditions. These errors are further studied in Section 6, which analyzes the magnitude of the errors and the image quality versus rendering time tradeoffs in addressing them.

4. An Anti-Aliasing Technique for Splatting

In this section we describe our splatting-based anti-aliasing method and argue for its correctness. In Section 4.1 we describe why anti-aliasing is needed for volume rendering algorithms. In Section 4.2 we develop an expression (Equation 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 4.3 we describe our anti-aliasing method, and in Section 4.4 we show that our method satisfies the equation developed in Section 4.2, which argues for the correctness of the method. Section 4.5 describes the anti-aliasing method’s frequency domain characteristics and discusses the effects of using a non-ideal reconstruction kernel.

4.1 The Need for Anti-Aliasing in Volume Rendering

The process of volume rendering is based on the integration (or composition), along an *integration grid*, of the volume raster. This integration grid is composed of *sight projectors* (or *rays*) which pass from the eye point, through the view plane, and into the volume raster. Before this integration can occur, the volume raster has to be reconstructed and then resampled along the integration grid. This is illustrated in Figure 3 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 pixels, which traverse the volume raster. Figure 3a 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 $+z$). The perspective transformation means the integration grid diverges as it traverses the volume. Figure 3b shows the same scene in *perspective space*, after perspective transformation and perspective division (perspective space is also commonly called *screen space*). Here the volume raster is distorted according to the perspective transformation, and because the eye is considered to be located at negative infinity, the integration grid lines are parallel.

The reconstruction and resampling of the volume raster onto the integration grid has to be done properly to avoid aliasing artifacts. Ideally, aliasing is avoided by the following criteria:

Criteria 1: Sample above the Nyquist limit.

Criteria 2: Reconstruct with an ideal filter.

The aliasing that results from an insufficient sampling rate (below the Nyquist limit) is called *prealiasing* (or sometimes *sample-rate aliasing*) — it is caused by energy from the alias spectra spilling over into the primary spectrum (see Figure 18i). The aliasing that results from a non-ideal reconstruction filter is called *postaliasing* — it is caused by the non-ideal filter picking up energy from the alias spectra beyond the Nyquist limit (see Figure 18b and [22,15,32]). In practice, it is not possible to implement an ideal reconstruction filter, and so Criteria 2 cannot be achieved — any realizable filter inevitably results in a tradeoff among aliasing, blurring, and other artifacts [15]. However, reconstruction filters previously used for splatting, in particular Gaussian filters, contribute very little postaliasing at the cost of substantial blurring [15]. In current splatting implementations, the great majority of aliasing is prealiasing; it arises from not achieving Criteria 1. Therefore, in the rest of this paper, when referring to the term ‘aliasing’ we generally mean prealiasing, or aliasing that results from not sampling above the Nyquist limit.

It may be possible to sample above the Nyquist limit, but if this is not possible then aliasing can also be avoided by low-pass filtering the volume to reduce its frequency content. For an orthographic view this low-pass filtering must be applied to the entire volume, but for a perspective view low-pass filtering may only be required for a portion of the volume. This can most easily be seen in perspective space (Figure 3b). Note that there is a distance along the $+z_p$ axis, denoted k_p, where the sampling rate of the volume raster and the sampling rate of the integration grid are the same. Previous to this distance there is less than one voxel per pixel, and beyond this distance there is more than one voxel per pixel. When there is more than one voxel per pixel, the volume raster can contain frequency information which is higher than the integration grid can represent, and so aliasing is possible. In the next section this concept is developed into an equation.

Note that same concept holds in eye space (Figure 3a). Here there is an equivalent distance along the $+z$ axis, denoted k, where the sampling rates of the volume raster and the integration grid are the same (note that in general $k \neq k_p$, but the two distances are related by the perspective transformation). Aliasing artifacts can occur beyond k, when the distance between adjacent rays is greater than one voxel.

Volume ray-casting algorithms generally perform the reconstruction in eye space. Some avoid aliasing by employing reconstruction kernels which become larger as the rays diverge [21,23]. This provides an amount of low-pass filtering which is proportional to the distance between the rays. Splatting algorithms generally perform the reconstruction in perspective space. Unlike ray-casting algorithms, existing splatting algorithms do not have an equivalent mechanism to avoid aliasing. In this paper we propose such a mechanism.

4.2 Necessary Conditions to Avoid Aliasing

In this section we give the conditions which are necessary for the volume rendering resampling process to avoid introducing sample-rate aliasing artifacts into the integration grid samples. Let *s* be the volume raster grid spacing (Figure 3a), and $\rho = 1/s$ be the volume raster sampling rate. The volume raster contains aliasing when either (1) the sampled function is not bandlimited, or (2) the function is bandlimited at frequency ω but the sampling rate ρ is below the Nyquist limit: $\rho < 2\omega$.

If the first condition holds then aliasing will be present no matter how large ρ becomes. However, if the function is bandlimited at ω, then as long as

$$\rho \geq 2\omega \tag{1}$$

there is no aliasing in the volume raster. Assuming that Equation 1 is true, our job is to resample the volume raster onto the integration grid in a manner that guarantees that no aliasing is introduced.

Let ϕ represent the sampling frequency of the integration grid. For a perspective projection the integration grid diverges (Figure 3a) and therefore ϕ is a function of distance along the z axis: $\phi = \phi(d)$, where *d* is the distance. For an orthographic projection we can still express $\phi(d)$ as a function, but it will have a constant value. As illustrated in Figure 3, at *k* the sampling rates of the volume raster and the integration grid are the same: $\rho = \phi(k)$.

The distance *k* naturally divides world space (and likewise k_p divides perspective space) into the following two regions (note that depending on the viewing parameters, the volume raster might lie entirely in either region; Figure 3 shows the most general case where the volume raster straddles the regions):

Region 1: $d < k$. This is the case for the region in Figure 3a previous to *k*. Here $\phi(d) > \rho$, and if Equation 1 holds then $\phi(d) > 2\omega$ and there is no sample-rate aliasing.

Region 2: $d \geq k$. This is the case for the region in Figure 3a beyond *k*. Here $\phi(d) \leq \rho$, and so it may be that $\phi(d) < 2\omega$. If this is the case, then the integration grid will contain a sample-rate aliased signal once *d* is large enough.

This argument shows that, given Equation 1, with a perspective projection volume rendering algorithms do not have to perform anti-aliasing within Region 1 (previous to the distance *k*). However, in Region 2 (beyond *k*), it is necessary to low-pass filter the volume raster to avoid aliasing. The amount of filtering required in Region 2 is a function of *d*, and should reduce the highest frequency in the volume raster from ω to $\tilde{\omega}(d)$ so that

$$\phi(d) \geq 2\tilde{\omega}(d) \tag{2}$$

in Region 2. To avoid aliasing with an orthographic projection the same equation must hold, although $\phi(d)$ and $\tilde{\omega}(d)$ will be constant functions for all values of *d*.

Regardless of the projection, by showing that Equation 2 holds for a particular volume rendering technique, we can claim that the technique does not introduce sample-rate aliasing artifacts when resampling from the volume raster onto the integration grid.

4.3 An Anti-Aliasing Method for Splatting

As mentioned in Section 4.1 above, volume ray-casting algorithms avoid aliasing by using reconstruction kernels which increase in size as the integration grid rays diverge, which satisfies Equation 2. In this section we give a similar anti-aliasing algorithm for splatting.

As shown in Figure 3a, at distance *k* the ratio of the volume raster sampling frequency ρ and the integration grid sampling frequency $\phi(d)$ is one-to-one. *k* can be calculated from similar triangles:

$$k = s\frac{D}{p}, \tag{3}$$

where *s* is the sample spacing of the volume raster, *p* is the extent of a pixel, and *D* is the distance from the eye point to the view plane.

Figure 4 gives a side view of “standard” splatting as implemented by Westover [29,30,31], as well as the new anti-aliasing method. In Figure 4 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. As in Figure 3, *D* is the distance of the view plane from the eye point, and *k* is calculated from Equation 3. For this example we are rendering a single row of splats, which are equally spaced along the z axis (Figure 4a). Because 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. Figure 4b shows the same scene in perspective space. Here D_p and k_p are *D* and *k* expressed in z_p coordinates. As expected, because of the non-linear perspective transformation, the splat spacing is now non-uniform along the z_p axis, and the size of the splats decreases with increasing distance from the eye.

The bottom row illustrates the anti-aliasing method. Previous to *k* we draw splats the same size in eye space (Figure 4c). Beginning at *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 Equation 4 below. Figure 4d shows what happens in perspective space. Previous to k_p we draw the splats with decreasing sizes according to the perspective transformation. Beginning at k_p, we draw all splats the same size, so splats with a z_p coordinate greater than k_p are the same size as splats with a z_p coordinate equal to k_p.

Figure 5 gives the geometry for scaling splats drawn beyond *k*. If a splat drawn at distance *k* has the radius r_1, then the radius r_2 of a splat drawn at distance $d > k$ is the projection of r_1 along the viewing frustum. This is calculated by similar triangles:

$$r_2 = \frac{d}{k} r_1. \tag{4}$$

Scaling the splats drawn beyond *k* is not enough to provide anti-aliasing, however. In addition to scaling, the energy that these splats contribute to the image needs to be the same as it would have been if they had not been scaled. As shown in Figure 5, splat 1 projects to the area S_1,and if splat 2 has not been scaled then it projects to the smaller area S_2. Because they are composited into the view plane in the form of two-dimensional “footprint” filter kernels [30], the amount of energy the splats contribute to the view plane is proportional to their areas (examples of energy measures include the volume under the splat kernel or the alpha channel of the polygon defining the 2D splat footprint; splat energy tends to vary according to the material properties of each voxel). Let E_1 and E_2 represent some energy measure for the splats. Because $S_1 \sim E_1$ and $S_2 \sim E_2$, by proportionality the following relationship holds between the splats’ energies and the splats’ projected areas:

$$E_2 = \frac{S_2}{S_1} E_1. \tag{5}$$

Note that Equation 5 follows from the assumption that splat 2 is not scaled. When splat 2 is scaled both splats project to the same sized area, S_1, on the view plane and hence contribute the same amount of energy. This is not what we want, and so we calculate splat 2’s energy by attenuating splat 1’s energy as shown in Equation 5 (or more precisely, we calculate splat 2’s energy by attenuating the energy splat 2 would have had if it had been positioned at distance *k*).

Now we transform Equation 5 into a form that is more computationally convenient. As shown in Figure 5, let A_1 and A_2 be the areas of the splats in world space. By similar triangles we can project the area S_2 to the area A_1: $A_1 / d^2 = S_2 / D^2$, which we can express as

$$A_1 = \left(\frac{d}{D}\right)^2 S_2. \tag{6}$$

With a similar argument we can project S_1 to A_2:

$$A_2 = \left(\frac{d}{D}\right)^2 S_1. \tag{7}$$

Then dividing Equation 6 by Equation 7:

$$\frac{A_1}{A_2} = \frac{S_2}{S_1}, \tag{8}$$

and substituting Equation 8 into Equation 5:

$$E_2 = \frac{A_1}{A_2} E_1. \tag{9}$$

Assume for now that the filter kernel is a circle for both splats. Then the areas of the splats are $A_1 = \pi r_1^2$ and $A_2 = \pi r_2^2$. By Equation 4 we can express A_2 in terms of r_1:

$$A_2 = \pi\left(r_1 \frac{d}{k}\right)^2. \tag{10}$$

Then

$$E_2 = \left(\frac{\pi r_1^2}{\pi\left(r_1 \frac{d}{k}\right)^2}\right) E_1 \tag{11}$$

which can be simplified to

$$E_2 = \left(\frac{k}{d}\right)^2 E_1. \tag{12}$$

Note that although here we have derived Equation 12 using circular filter kernels, we can also use any other two-dimensional shape for the kernel, such as an ellipse, square, rectangle, parallelogram, etc. and derive the same equation. We use Equation 12 to attenuate the energy of all splats drawn beyond *k*.

4.4 Correctness of the Method

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

We derive the integration grid sampling frequency $\phi(d)$ with a similar-triangles argument. Consider Figure 6, where $q(d)$ is the integration grid spacing at distance *d* from the eyepoint. By similar triangles

$$\frac{s}{k} = \frac{q(d)}{d}, \tag{13}$$

which can be written as

$$\frac{1}{q(d)} = \frac{1}{s} \cdot \frac{k}{d}. \tag{14}$$

Now $1/s = \rho$, and $1/q(d)$ is simply $\phi(d)$, the integration grid sampling frequency at *d*. Thus we have

$$\phi(d) = \rho \cdot \frac{k}{d}. \tag{15}$$

The maximum volume raster frequency $\tilde{\omega}(d)$ can be derived from the scaling property of the Fourier transform [1]:

$$f\left(\frac{1}{a}t\right) \leftrightarrow |a|F(a\omega), \tag{16}$$

where "↔" indicates a Fourier transform pair. This says that widening a function by the factor *a* in the spatial domain is equivalent to narrowing the function in the frequency domain by the factor $1/a$. In our anti-aliasing method, the widening for the splat kernels drawn beyond *k* is given by Equation 4:

$$a = \frac{d}{k}. \tag{17}$$

Thus we have

$$f\left(\frac{k}{d}t\right) \leftrightarrow \left|\frac{d}{k}\right|F\left(\frac{d}{k}\omega\right), \tag{18}$$

which shows that as the splat kernels are widened by d/k, the frequency components of the function they reconstruct are narrowed by k/d. Since all the frequencies of the volume raster are attenuated by k/d, the maximum frequency ω is attenuated by the same amount, and we have:

$$\tilde{\omega}(d) = \omega \cdot \frac{k}{d}. \tag{19}$$

Now we are ready to show that our technique satisfies Equation 2. We start with Equation 1: $\rho \geq 2\omega$, which implies that the volume raster has sampled the function above the Nyquist limit. Multiplying both sides by k/d we have

$$\rho \cdot \frac{k}{d} \geq 2\left(\omega \cdot \frac{k}{d}\right), \tag{20}$$

which we can write as:

$$\phi(d) \geq 2\tilde{\omega}(d). \tag{21}$$

This derivation says that if the volume raster has sampled the function above the Nyquist limit, our anti-aliasing 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 non-ideal reconstruction filter.

4.5 Frequency and Spatial Domain Behavior

This section describes the frequency and spatial domain behavior of the anti-aliasing method, and it illustrates the effects of reconstructing with a non-ideal reconstruction kernel. Figures 18 and 19 compare the standard splatting technique with the new anti-aliasing technique. The figures analyze the technique in two planes which are parallel to the view plane and include the volume raster and integration grid, where one plane is located previous to the distance *k* and the other plane is located beyond the distance *k*. For clarity, the diagrams are all drawn in one dimension; the extension to two dimensions is straightforward.

Figure 18a shows the sampled signal in the frequency domain. Here the spectra have maximum frequency ω and are replicated at intervals of ρ. Assuming Equation 1 is true, $\rho \geq 2\omega$ and there is no prealiasing in the sampled signal. Figure 19a shows the sampled signal in the spatial domain, with sample spacing *s*.

The left-hand column of both figures shows the behavior of splatting previous to *k*. Figure 18b shows an ideal splatting kernel, which has the value 1 in the range $-(1/2)\rho$ to $(1/2)\rho$ and 0 elsewhere, and yields only the primary spectrum. This is compared to a realizable splatting kernel of radius R_1, which has some smoothing in the passband and some postaliasing in the stop band [15, 22]. Figure 19b shows the same two kernels in the spatial domain, where the realizable kernel has the radius r_1. The sampled signal is multiplied (convolved) by the splatting kernel, yielding a reconstructed signal (Figures 18c and 19c). This signal is then resampled by the integration grid samples (Figures 18d and 19d), which have a higher frequency than the volume raster. Figures 18e and 19e show the final result: the reconstructed signal is convolved (multiplied) by the integration grid samples. Because the spectra are farther apart than in the original signal (Figure 18a), no pre-aliasing is introduced.

The middle column of both figures shows the behavior of the standard splatting implementation beyond *k*. The same splatting kernel used previous to *k* is used here (Figures 18f and 19f), yielding the same reconstructed signal (Figures 18g and 19g). However, now the integration grid samples have a lower frequency than the volume raster (Figures 18h and 19h). Figure 18i shows the result in the frequency domain: the spectra of the resampled signal, although identical to those in Figure 18e, are pulled closer together and overlap, resulting in prealiasing. In the spatial domain (Figure 19i), the integration grid samples are not spaced close together enough to capture the variation in the reconstructed signal.

The right-hand column of both figures shows the behavior of the anti-aliased splatting technique beyond *k*. Figure 18j shows that in the frequency domain, the ideal splatting kernel would no longer pass all the frequency content in the volume raster ($-(1/2)\rho$ to $(1/2)\rho$), but would instead only pass the frequency content that could be represented by the integration grid ($-(1/2)\phi(d)$ to $(1/2)\phi(d)$). This is exactly what happens when modulating the reconstruction kernel by k/d (Equation 15). As shown in Figure 18j, the splatting kernel used beyond *k* is also narrowed by k/d, giving it the radius

$$R_2 = \frac{k}{d}R_1. \tag{22}$$

In the spatial domain the splatting kernel has radius r_2 (Figure 19j), as given by Equation 4. The resulting reconstructed signal has less high-frequency content (Figure 18k), and is blurred in the spatial domain (Figure 19k). When convolved by the integration grid samples (Figures 18l and 19l), the spectra of the resampled signal do not overlap (Figure 18m). In the spatial domain, the integration grid samples (Figure 19l) capture the dampened variation in the resampled signal (Figure 19m).

Thus, the prealiasing evident in the middle column is prevented in the right column. Although preventing the aliasing comes at the cost of additional low-pass filtering, the filtering is applied as a function of *d*, and so it only removes frequency content that cannot be represented by the integration grid at distance *d*.

5. Results

In our implementation of this algorithm we make use of rendering hardware to quickly draw the splats, in a manner similar to [5], [19], and [35]. For each splat we draw a polygon in world space centered at the voxel position. The polygon is rotated so it is perpendicular to the ray passing from the eye point through the voxel position. The splat kernel is pre-computed 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 [5]. We attenuate the alpha channel of the polygon as a measure of splat energy when evaluating Equation 12, and then composite the semi-transparent splat polygon into the screen buffer.

Our renderer is a modified version of the “splat renderer” reported in [35]. 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.

Figures 7 shows a $498 \times 398 \times 1$ volume consisting of a single sheet, where alternate 10×10 squares are colored either red or white to create a checkerboard effect. The resulting dataset contains 198K splats which are rendered into a 270×157 image. In Figure 7a a black line is drawn at the distance k; beyond this line there is more than one voxel per pixel. As expected, the upper image shows strong aliasing effects, but these are smoothed out in the lower image.

Figure 8 shows a $512 \times 512 \times 103$ volume containing a terrain dataset acquired from a satellite photograph and a corresponding height field. The resulting dataset contains 386K splats (this is more than the expected 512^2 splats because extra splats are required to fill in the “holes” formed where adjacent splats differ in height). Each column of splats is given the color of the corresponding pixel from the satellite photograph. The dataset is rendered into a 270×175 image. In Figure 8a a black line is drawn at the distance k. The upper image shows strong aliasing in the upper half of the image (containing about 90% of the data); when animated, these regions show prominent flickering and strobing effects. In the lower image these regions have been smoothed out; and although our anti-aliasing technique does not directly address temporal aliasing effects, when animated these regions are free of flickering and strobing effects.

Figure 9 explores this temporal anti-aliasing in greater detail. The figure shows how a one-pixel-wide column varies over time. To create Figure 9, a rectangular region one pixel wide by 50 pixels high was taken from successive frames of an animation of the terrain dataset (Figure 8 is a single frame from this animation). These rectangles were abutted to form the columns of Figure 9, and then each pixel was expanded into a 5×5 block. Thus, the rows of Figure 9 are a spatial representation of the temporal progression of the animation. Figure 9a shows the frames from the aliased dataset (Figure 8a), while Figure 9b shows the frames from the anti-aliased dataset (Figure 8b). Note the almost random pixel colors that occur across the rows of Figure 9a — these represent the strong flickering and strobing artifacts that are present in the animation. In comparison, the rows of Figure 9b show smooth variations of color, and indicate the absence of these temporal artifacts.

Figure 10 shows a $420 \times 468 \times 62$ volume containing a microtubule study acquired from confocal microscopy. The resulting dataset contains 103K splats which are rendered into a 270×270 image.

Unlike the previous images, where the viewpoint is set so that the datasets disappear into the horizon and thus the splats have a great range of sizes (covering several pixels to much less than a single pixel), in this image the entire dataset is visible, and the range of splat sizes is much smaller. However, because the volume has a higher resolution than the image it is still liable to aliasing effects (all of the splats are drawn beyond the distance *k*). This is shown in the upper image, which contains jagged artifacts that shimmer when animated. In the lower image these effects have been smoothed out; when animated this shimmering effect disappears.

Figure 11 explores the temporal anti-aliasing of the animated version of Figure 10 — it shows how a one-pixel-wide column varies over time, and was created by the same process used to create Figure 9. As in Figure 9, Figure 11a shows the aliased frames, while Figure 11b shows the anti-aliased frames. The microtubule dataset clearly contains less high-frequency content than the terrain dataset (Figure 8), and consequently the microtubule animation displays less temporal aliasing. However, there is still a definite and discernible "crawling ant" effect surrounding the edges of the microtubule structures. This is depicted in the rows of Figure 11a by the sudden change between the black background and the grey voxels — in Figure 11b the transitions between black and grey are much smoother.

6. Analysis of Perspective Splatting Errors

The original splatting formulation by Westover [29] was only fully implemented for orthographic views. In part this was done for efficiency reasons: for an orthographic view, only one footprint function is needed — it can be computed once and translated to the projection point of every voxel in the volume. As discussed in Section 3.3 above, when used to render perspective views the original splatting formulation has two implementation inaccuracies:

(1) the footprint polygon is not aligned perpendicularly with the sight ray passing through the polygon center, and

(2) every sight ray coming from every pixel does not traverse the footprint polygon at a perpendicular angle.

The first condition is easy to remedy in a perspective splatting implementation, but at the cost of additional operations per voxel. To date there is not a known way to address the second condition, outside of the ray-driven splatting method [19], which casts rays into the volume and uses splatting kernels to reconstruct the function at ray sample points.

This section analyzes the errors caused by these two implementation inaccuracies. It gives recommendations for when the first condition might be tolerated in the interest of faster rendering, and it quantifies the errors caused by the second condition.

6.1 Align Footprint Polygon Perpendicular to Center Ray

Figure 12 compares the projection of a polygon of extent $2 \cdot ext$ onto the image plane (located at x_s on the viewing axis). We illustrate both the correct case, where the footprint polygon is perpendicular to the center ray, and the approximate case, in which the footprint polygon is aligned with the voxel grid. The correct case requires that each polygon be translated to the voxel position (x_v, y_v), and then rotated to be perpendicular to the center ray. For the approximate case, each polygon only requires a translation. Thus the approximate case can save many operations per voxel, and makes for a faster (but less accurate) splatting implementation.

From Figure 12 we observe that the projected polygon extent Δy_{approx} in the approximate case is slightly smaller than the projected polygon extent Δy_{corr} in the correct case. Using simple trigonometric arguments one can show that

$$\Delta y_{corr} = \frac{2 \cdot ext \cdot x_s (y_v \sin\varphi_c - x_v \cos\varphi_c)}{x_v^2 - ext^2 \cdot \sin\varphi_c^{\,2}}, \tag{23}$$

where φ_c is the viewing frustum half-angle. One can also show that

$$\Delta y_{approx} = \frac{2 \cdot ext \cdot x_s}{x_v}. \tag{24}$$

Thus

$$\Delta y_{corr} = \Delta y_{approx} \cdot \frac{x_v^2}{\cos\varphi_c (x_v^2 - ext^2 \cdot \sin\varphi_c^{\,2})} \approx \Delta y_{approx} \cdot \frac{1}{\cos\varphi_c}, \tag{25}$$

since $x_v^2 \gg ext^2 \cdot \sin\varphi_c^{\,2}$. This means that the error is largest at the boundary of the viewing frustum. For a viewing frustum half-angle φ_c= 30°, this scaling factor is 1.15.

The absolute error Δy_{approx} - Δy_{corr} between the correct and the approximate screen projection of a kernel with extent *ext* is given by

$$\begin{aligned} \Delta y_{corr} - \Delta y_{approx} &= \frac{x_v^2 - \cos\varphi_c (x_v^2 - ext^2 \cdot \sin\varphi_c^{\,2})}{x_v \cos\varphi_c (x_v^2 - ext^2 \cdot \sin\varphi_c^{\,2})} \\ &\approx \frac{2 \cdot ext \cdot x_s (1 - \cos\varphi_c)}{x_v \cos\varphi_c} = 2 \cdot ext \cdot \frac{x_s}{x_v}\left(\frac{1}{\cos\varphi_c} - 1\right). \end{aligned} \tag{26}$$

This error is plotted in Figure 13 for a volume of 256^3, a viewing frustum half-angle φ_c=30°, and a kernel extent of *ext*=2.0. From this plot we see that voxels close to the viewplane and at the edge of the viewing frustum suffer rather large errors: the maximum error is 0.36 times the kernel size. However, voxels far away from the viewplane and close to the viewing axis cause only small absolute errors.

Thus we need to consider this error only for objects close to the viewing frustum boundary *and* close to the viewplane. Hence, we may want to use translated and rotated footprint polygons for voxels closer to the eyepoint, and only translated polygons for all other voxels. Note that these errors are not only committed when mapping the extent of the polygon, but also when mapping the inside of the polygon (e.g. the footprint lookup table) to the screen.

6.2 Align Every Sight Ray Perpendicular to Footprint Polygon

Figure 14 shows the 2D case of the footprint polygon aligned perpendicularly to the center ray. Pixel rays other than the center ray traverse the footprint polygon at an oblique angle and should have the footprint polygon aligned as depicted by the dotted line. The error when computing the lookup table index *t* for these rays is given by

$$t_{corr} = t_{approx} \cdot \cos\varphi, \tag{27}$$

where t_{corr} is the correct index and t_{approx} is the computed index. The angle φ is largest for voxels close to the eye and at the center of the image plane, where for a volume of 256^3, a viewing frustum half-angle φ_c=30°, and a kernel extent of *ext*=2.0, the approximation t_{corr}/t_{approx}=0.99 at the polygon boundary is rather good. Unlike the footprint polygon alignment problem discussed above, there is not a know solution to this problem.

7. Future Work

7.1 Applications to Texture Mapping

The anti-aliased splatting technique described in this paper can also perform traditional texture mapping of surface-based geometric primitives such as polygons and patches. This section demonstrates that the technique provides an accurate solution to the texture sampling problem — one that is more accurate than either mip-maps [33] or summed area tables [6].

Recall that traditional texture mapping algorithms transform a pixel (or the filter kernel) from screen into texture space, using the inverse of the viewing transformation (Figure 15). In general, the preimage of a square pixel is an arbitrary quadrilateral in texture space, while the preimage of a circular filter kernel is a conic curve [10]. The highest quality (and most expensive) texture mapping techniques integrate the texture map under this preimage [8, 9], which gives the color that is applied to the pixel. While these texture mapping methods perform very high-quality texture mapping, they are very expensive and are seldomly used. There are at least two reasons why they are so expensive: 1) each pixel must be inverse transformed, and 2) the texture map must be accessed many times, and this access is incoherent. This second point is illustrated in Figure 16, which shows what can happen when two adjacent pixels are texture mapped. In this example, the preimages of the two pixels overlap in texture space, which means all the texture elements in the overlapped area must be accessed twice.

Most current texture mapping implementations use mip-maps [28]. As shown in Figure 15, a summed area table integrates the texture map in the axis-aligned rectangular bounding box of the pixel preimage, while a mip-map integrates the texture map in the axis-aligned square bounding box. This means that both methods integrate over a larger area than is required, and hence they perform more low-pass filtering than is actually needed, resulting in a texture that is more blurry than necessary.

As shown by Figures 7 and 8, the anti-aliasing method presented in this paper can also be applied to texture mapping — Figures 7 and 8 are simply volumetric objects, and any 2D or 3D texture map can likewise be considered a volumetric object. Using the new technique, demonstrated in Figure 17, every 2D or 3D texture sample is mapped according to the forward viewing transformation and composited into the image plane, using the variably-sized, attenuated filter kernel discussed above.

This technique has several advantages for texture mapping. First, the resulting filtering quality is equivalent to high-quality methods, because each pixel only receives the contribution of those texture samples that lie in the pixel's preimage. In particular, the filtering is of higher quality than either mip-mapping or summed area tables. Second, the method is less expensive than methods which integrate under the pixel preimage, because 1) each texture sample is forward transformed, which is less expensive than an inverse-transformation, 2) no texture sample has to be accessed more than once, and 3) the texture map is accessed coherently. A similar type of coherent texture access has been used in the REYES rendering architecture for performance reasons [4].

7.2 Other Areas of Future Work

Another area of future work is modifying our implementation to take better advantage of the rendering hardware. As reported in [35], depending on the machine and the number of extracted voxels, our renderer can give real-time performance. However, our current implementation does not take full advantage of the rendering speeds offered by the graphics hardware. Since the anti-aliased splats defined by Equation 4 project to the same size on the screen, it would be more efficient to draw the splats directly on the screen instead of drawing the scaled splats in eye space. We are currently looking into hardware-supported *bitblt* operations to provide optimal rendering speeds.

Another area of future work is an effort in the opposite direction. In this project there is a mismatch between the type of rendering algorithm we have written and the available programming tools: the

Silicon Graphics rendering hardware we used is designed to accelerate scenes using traditional surface graphics primitives such as polygons and spline surfaces; it does not contain an optimized splat primitive. To this end, new hardware architectures which better support operations which are common in volume rendering are needed. Some possible avenues of exploration are:

- Extended *bitblt*-like operators that can be sub-pixel centered and subsequently composited.
- Hardware support for point rendering using different reconstruction kernels (cubic, Gaussian, etc.) with common footprints (circular, elliptical, etc.). Potentially this offers a far more efficient implementation of splatting than hardware texture mapping.
- A higher resolution alpha channel to allow for the accurate accumulation of very transparent splats.
- Splat primitives with automatic size scaling based on their *z*-depth.

A splat primitive has properties of both simple points (always lying in the projection plane) and texture maps (non-linear intensities across the primitive). We are currently working on scanline algorithms to efficiently render splat primitives, with the goal of future hardware implementations. We are exploring this point-based approach to viewing and rendering as an alternative to the traditional scanline rendering of discrete objects, and we are exploring its applications for texture mapping, image-base rendering, and volume rendering.

8. Acknowledgments

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] Bracewell, R. N., *The Fourier Transform and Its Applications* (2nd edition), McGraw-Hill, 1978.

[2] Bentum, M. J., Lichtenbelt, B. B. A., Malzbender, T., “Frequency Analysis of Gradient Estimators in Volume Rendering”, *IEEE Transactions on Visualization and Computer Graphics*, 2(3), September 1996, pp. 242–254.

[3] Cabral, B., Cam, N., and Foran, J., “Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware”, In proceedings of *1994 Symposium on Volume Visualization* (Washington, DC, October 17–18), IEEE Computer Society Press, 1994, pp. 91–98.

[4] Cook, R. L., Carpenter, L., & Catmull, E. “The REYES Image Rendering Architecture”. *SIGGRAPH '87 Conference Proceedings*, 21(4), pp. 95-102, July 1987.

[5] Crawfis, R., and Max, N., “Texture Splats for 3D Scalar and Vector Field Visualization”, In

proceedings of *Visualization ’93* (San Jose, CA, October 25–29), IEEE Computer Society Press, 1993, pp. 261–266.

[6] Crow, F. C. “Summed-Area Tables for Texture Mapping”. *SIGGRAPH '84 Conference Proceedings*, 18(3), pp. 207-212., July 1984.

[7] Drebin, R. A., Carpenter, L., and Hanrahan, P., “Volume Rendering”, *Computer Graphics* (Proceedings of SIGGRAPH), 22(4), August 1988, pp. 65–74.

[8] Feibush, E. A., Levoy, M., and Cook, R. L. “Synthetic Texturing Using Digital Filters”. *SIGGRAPH '80 Conference Proceedings*, 14(3), pp. 294-301, July 1980.

[9] Greene, N., and Heckbert, P. S. “Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter”. *IEEE Computer Graphics and Animation*, 6(6), pp. 21-27, June 1986.

[10] Heckbert, P. S., *Fundamentals of Texture Mapping and Image Warping*. Masters Thesis, Department of Electrical Engineering and Computer Science, The University of California at Berkeley, Technical Report Number UCB/CSD 89/516, June 1989.

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

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

[13] Laur, D. and Hanrahan, P., “Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering,” *Computer Graphics* (Proceedings of SIGGRAPH), 25(4), July 1991, pp. 285–288.

[14] Machiraju, R. and Yagel, R., “Data-Parallel Volume Rendering Algorithms”, *The Visual Computer*, 11(6), 1995, pp. 319–338.

[15] Marschner, S.R. and Lobb, R.J., “An Evaluation of Reconstruction Filters for Volume Rendering”, In proceedings of *Visualization ’94* (Washington, DC, October 17–21), IEEE Computer Society Press, 1994, pp. 100–107.

[16] Mao, X., “Splatting of Non Rectilinear Volumes Through Stochastic Resampling,” *IEEE Transactions on Visualization and Computer Graphics*, 2(2), 1996, pp. 156–170.

[17] Max, N., “An Optimal Filter for Image Reconstruction,” In *Graphics Gem II*, J. Arvo, Editor, Academic Press, pp. 101–104.

[18] Möller, T., Machiraju, R., Mueller, K., and Yagel, R., “Classification and Local Error Estimation of Interpolation and Derivative Filters for Volume Rendering”, In proceedings of *1996 Symposium on Volume Visualization* (San Francisco, CA, October 27 – November 1), IEEE Computer Society Press, 1996, pp. 71–78.

[19] Mueller, K. and Yagel, R., “Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach”, In proceedings of *Visualization ’96* (San Francisco, CA, October 27 – November 1), IEEE Computer Society Press, 1996, pp. 65–72.

[20] Mueller, K., Yagel, R., and Wheller, J. J., “Accurate Low-Contrast 3D Cone-Beam Reconstruction With Algebraic Methods”, In submission to *IEEE Transactions on Medical Imaging*.

[21] Novins, K.L., Sillion, F.X., and Greenberg, D.P., “An Efficient Method for Volume Rendering using Perspective Projection”, *Computer Graphics* (Proceedings of San Diego Workshop on Volume Visualization), 24(5), November 1990, pp. 95–102.

[22] Oppenheim, A.V., and Shafer, R.W., *Digital Signal Processing*, Prentice-Hall, 1975.

[23] Pfister, H. and Kaufman, A., “Cube-4 — A Scalable Architecture for Real-Time Volume Rendering”, In proceedings of *Visualization ’96* (San Francisco, CA, October 27 – November 1), IEEE Computer Society Press, 1996, pp. 47–54.

[24] Reynolds R.A., Gordon D., and Chen L., “A Dynamic Screen Technique for Shaded Graphics Display of Slice-Represented Objects”, *Computer Vision, Graphics, and Image Processing*, 38(3), June 1987, pp. 275–298.

[25] Sobierajski L.M. and Avila R.S., “Hardware Acceleration for Volumetric Ray Tracing,” In proceedings of *Visualization ’95* (Atlanta, GA, October 29 – November 3), IEEE Computer Society Press, 1995, pp. 27–34.

[26] Swan II, J. E., Mueller, K., Möller, T., Shareef, N., Crawfis, R., and Yagel, R. “An Anti-Aliasing Technique for Splatting.” *Proceedings IEEE Visualization '97*, IEEE Computer Society Press, 1997, pp. 197-204.

[27] Swan, J. Edward II, *Object-Order Rendering of Discrete Objects*, Ph.D. Dissertation, Department of Computer and Information Science, The Ohio State University, 1997.

[28] Watt, A. and Watt, M. *Advanced Animation and Rendering Techniques, Theory and Practice.* Addison-Wesley Publishing Company, New York, New York, 1992.

[29] Westover, L.A., “Interactive Volume Rendering”, In proceedings of *Volume Visualization Workshop* (Chapel Hill, NC, May 18–19), Department of Computer Science, University of North Carolina, Chapel Hill, NC, 1989, pp. 9–16.

[30] Westover L.A., “Footprint Evaluation for Volume Rendering”, *Computer Graphics* (Proceedings of SIGGRAPH), 24(4), August 1990, pp. 367–376.

[31] Westover, L.A., *SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm*, Ph.D. Dissertation, Department of Computer Science, The University of North Carolina at Chapel Hill, 1991.

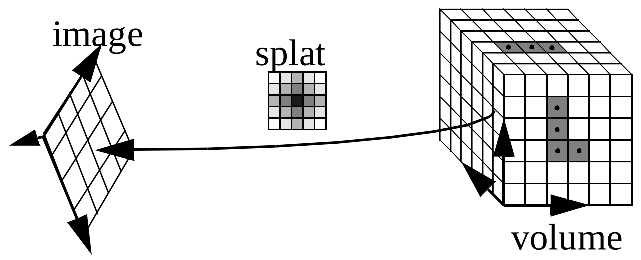
[32] Wolberg, G. *Digital Image Warping*. IEEE Computer Society Press., Los Alamitos, California, 1990.

[33] Williams, L. “Pyramidal Parametrics”. *SIGGRAPH '83 Conference Proceedings*, 17(3), pp. 1-11., July 1983., G., *Digital Image Warping*, IEEE Computer Society Press, 1989.

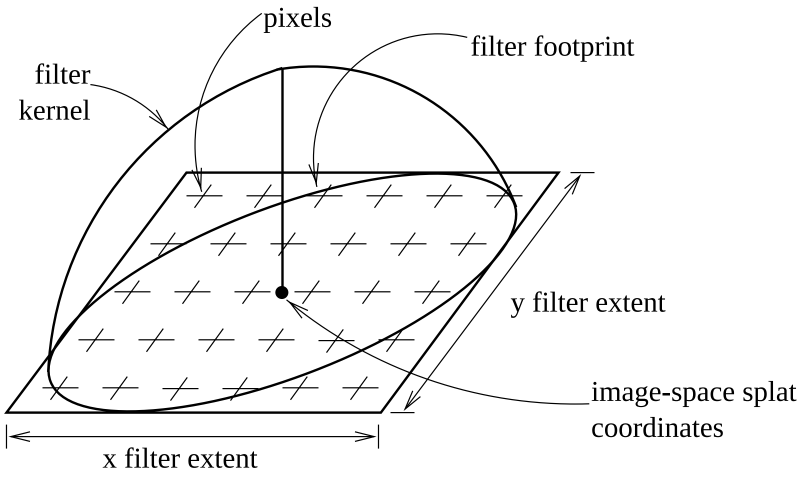
[34] Yagel, R., and Shi, Z., “Accelerating Volume Animation by Space-Leaping”, In proceedings of *Visualization ’93* (San Jose, CA, October 25–29), IEEE Computer Society Press, 1993, pp. 62–69.

[35] Yagel, R., Ebert, D.S., Scott, J., and Kurzion, Y., “Grouping Volume Renderers for Enhanced Visualization in Computational Fluid Dynamics,” *IEEE Transactions on Visualization and Computer Graphics*, 1(2), 1995, pp. 117–132.

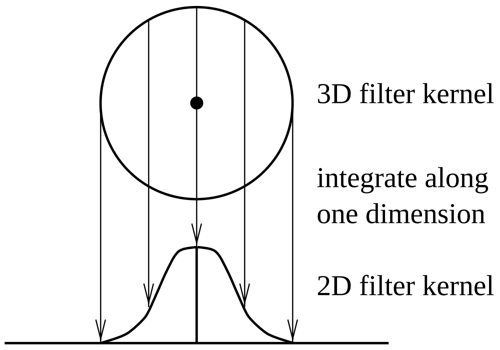
Figures and Tables



(a)



(b)



(c)

FIGURE 1. Overview of the splatting algorithm. (a) Projecting each voxel from the volume raster into the image plane. (b) Positioning each splat on the screen and compositing the contribution into each pixel. (c) The 3D rotationally symmetric filter kernel is integrated to produce a 2D filter kernel or “footprint table”.

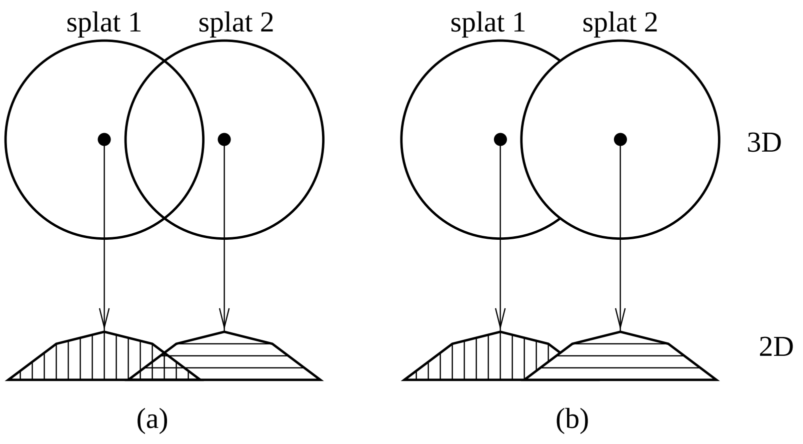
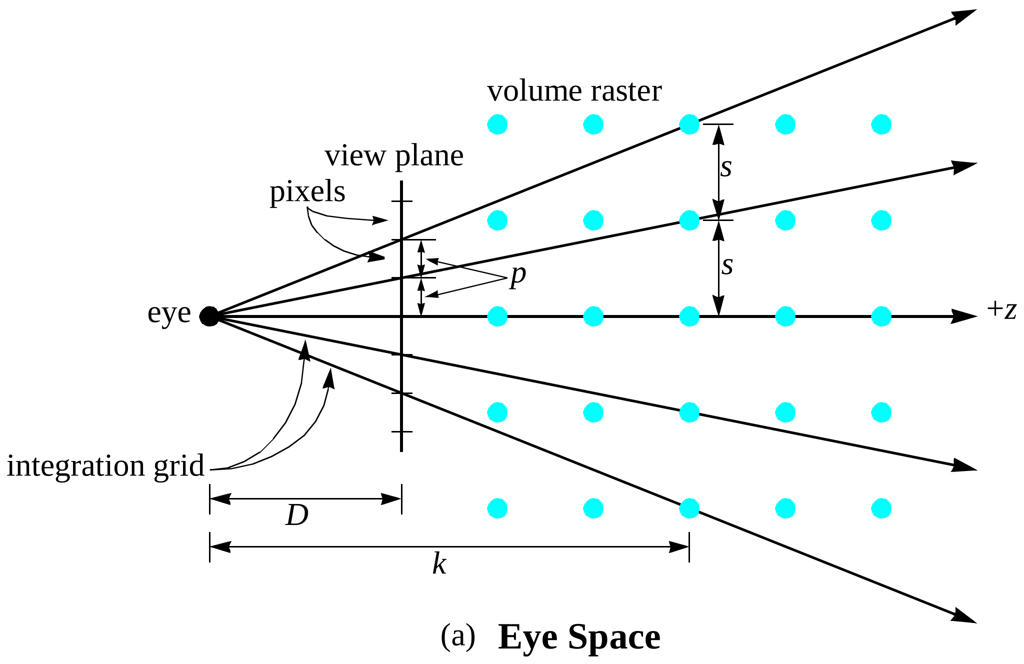


FIGURE 2. The overlapping splat problem. Assume that splat 1 is drawn before splat 2. (a) The desired result: the splat contributions are summed where they overlap. (b) The actual result: splat 2 is composited on top of splat 1, and splat 1 is occluded where the splats overlap.



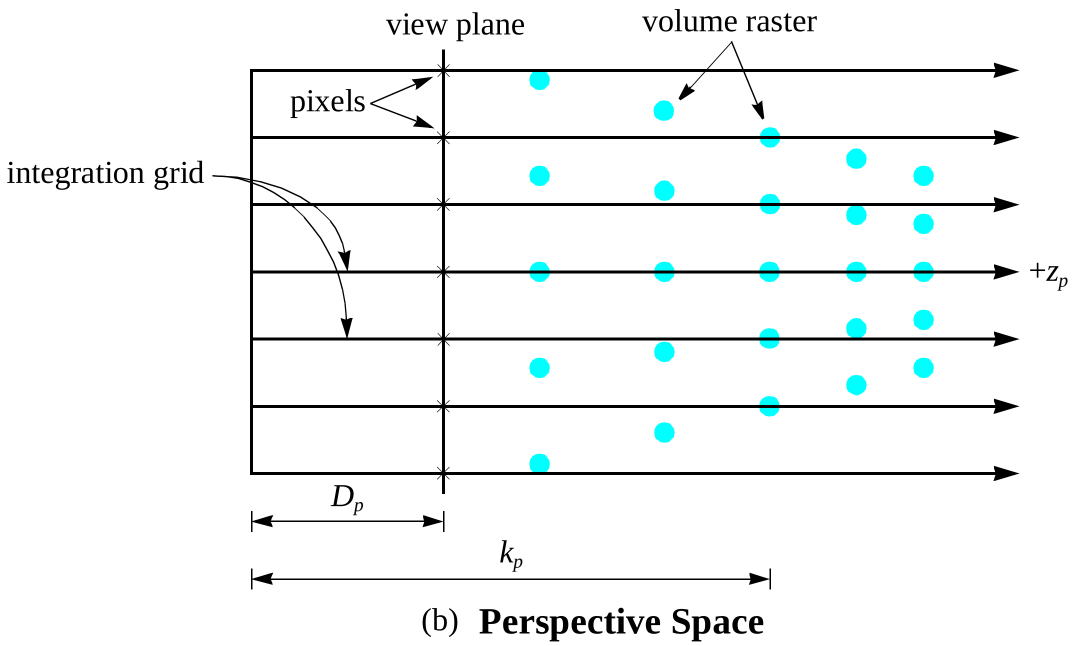


FIGURE 3. Resampling the volume raster onto the integration grid. (a) In eye space. (b) In perspective space.

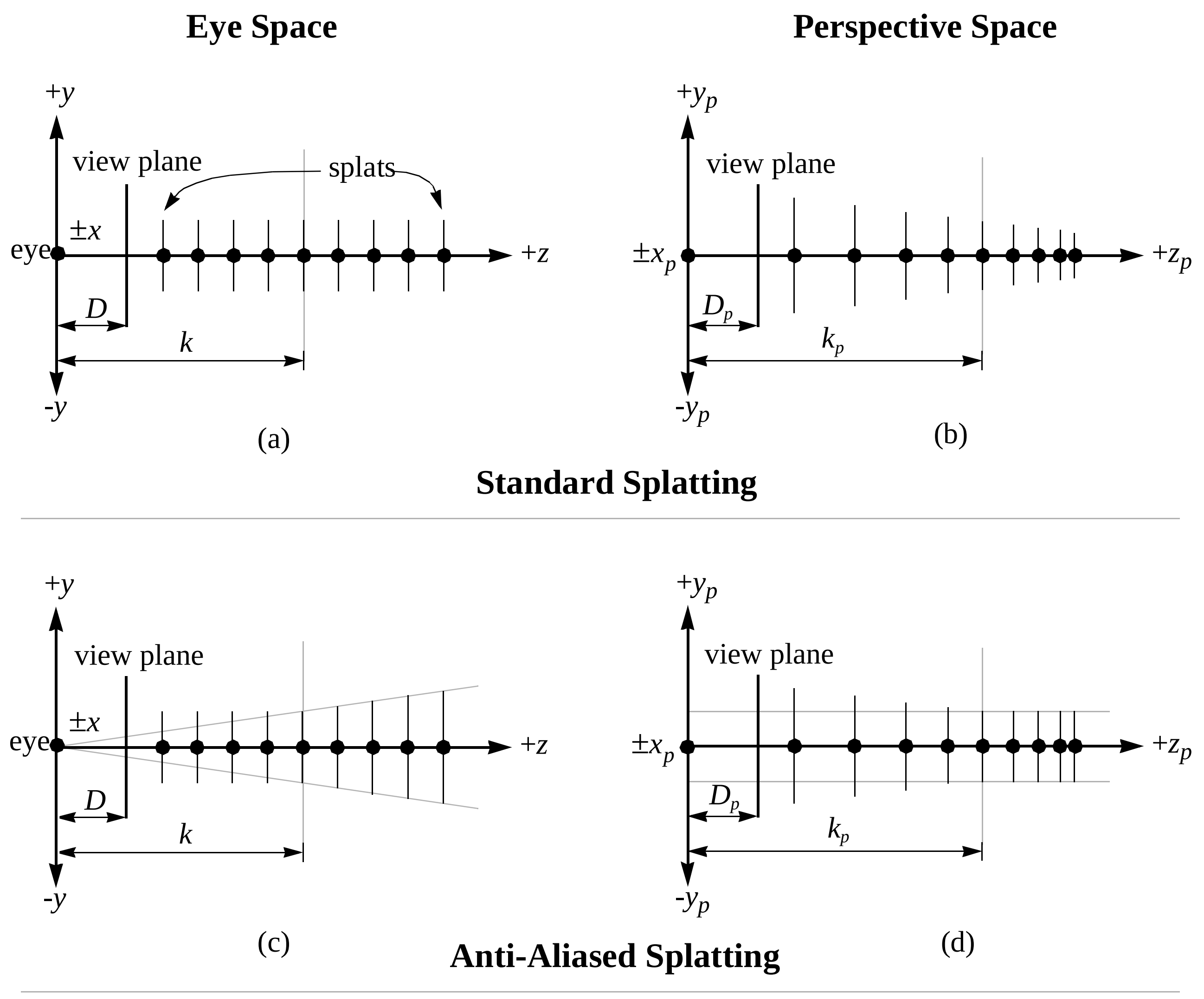


FIGURE 4. A comparison of the standard splatting method (top) with our anti-aliased method (bottom). (a) The standard splatting method in eye space. (b) The standard splatting method in perspective space. (c) The anti-aliased splatting method in eye space. (d) The anti-aliased splatting method in perspective space.

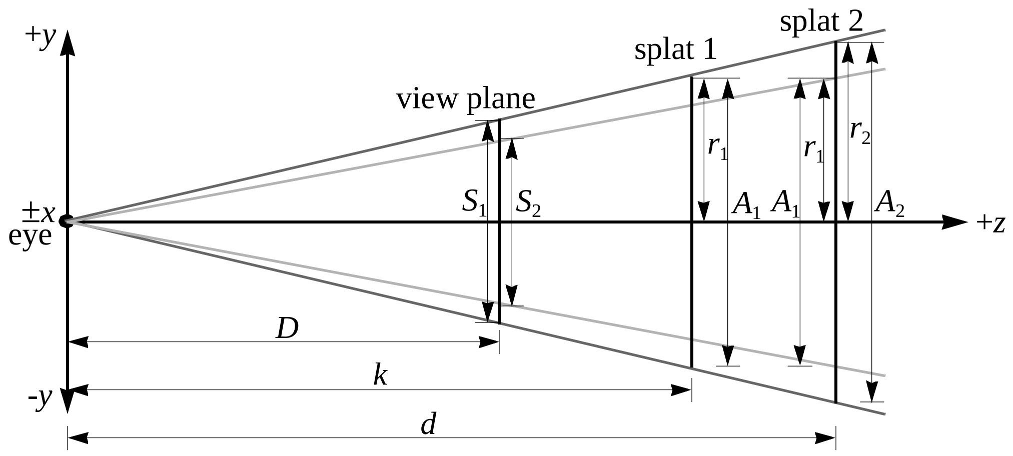


FIGURE 5. The geometry for attenuating the energy of splat 2.

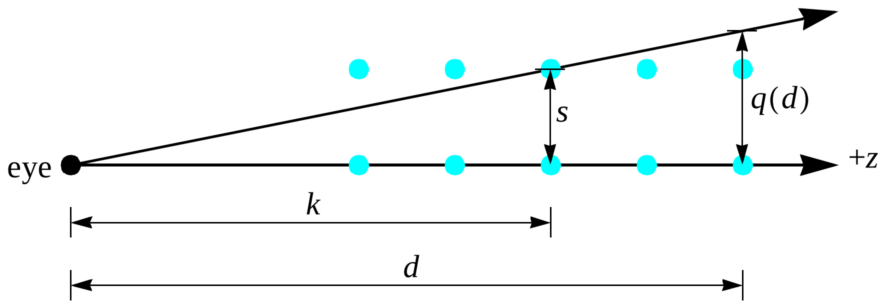
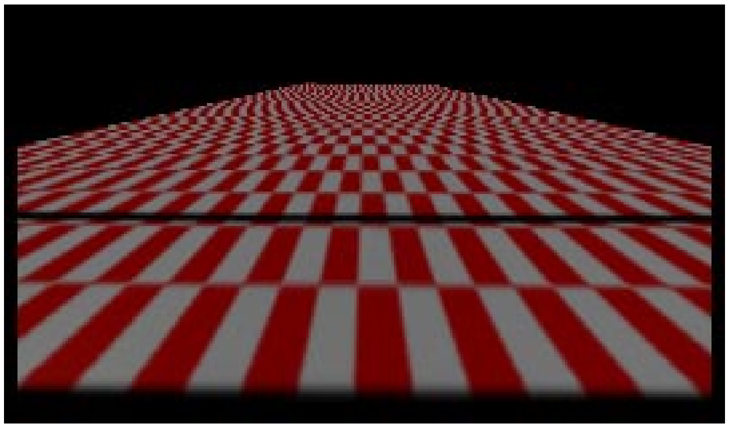
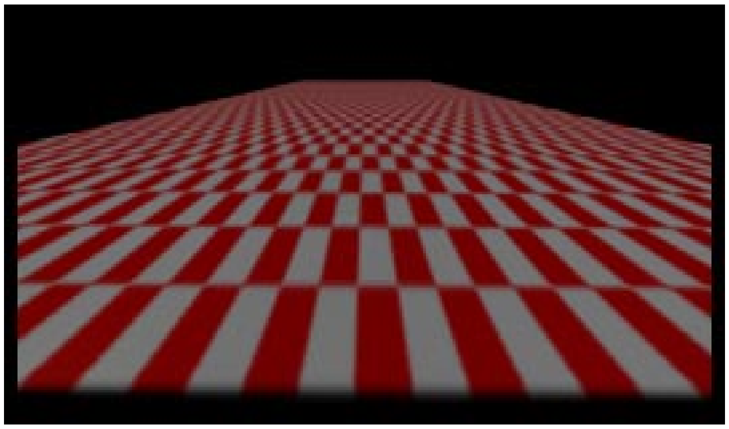


FIGURE 6. Calculating the integration grid sampling frequency.

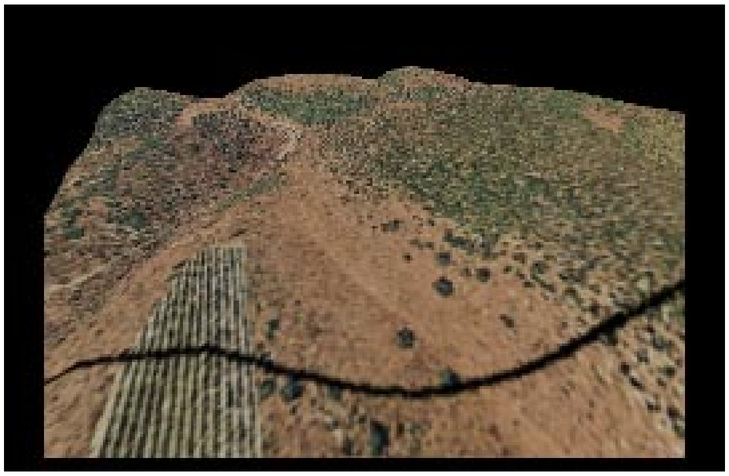


(a)

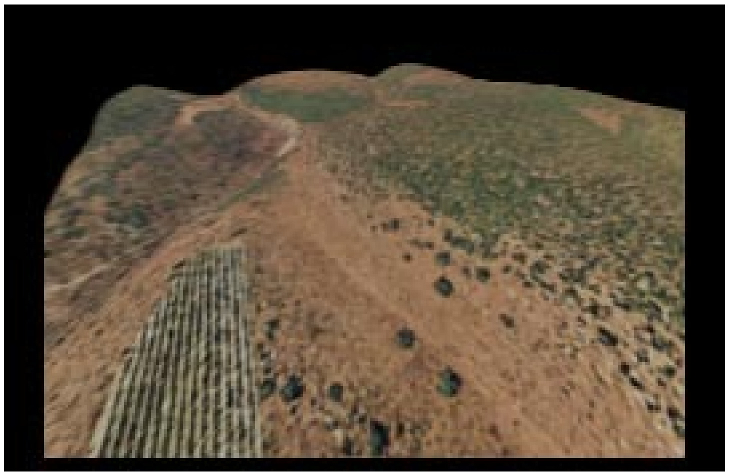


(b)

FIGURE 7. Rendered image of a plane with a checkerboard pattern. (a) Rendered with standard splatting; the black line is drawn at distance *k*. (b) Rendered with anti-aliased splatting.



(a)



(b)

FIGURE 8. Rendered image of a terrain dataset from satellite and mapping data. (a) Rendered with standard splatting; the black line is drawn at distance *k*. (b) Rendered with anti-aliased splatting.

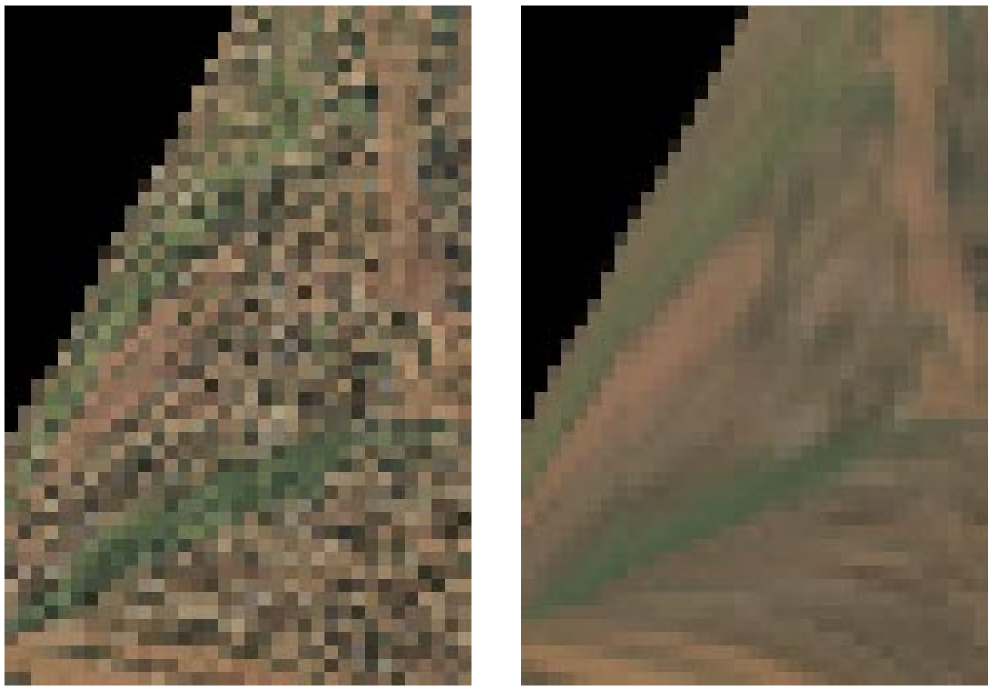
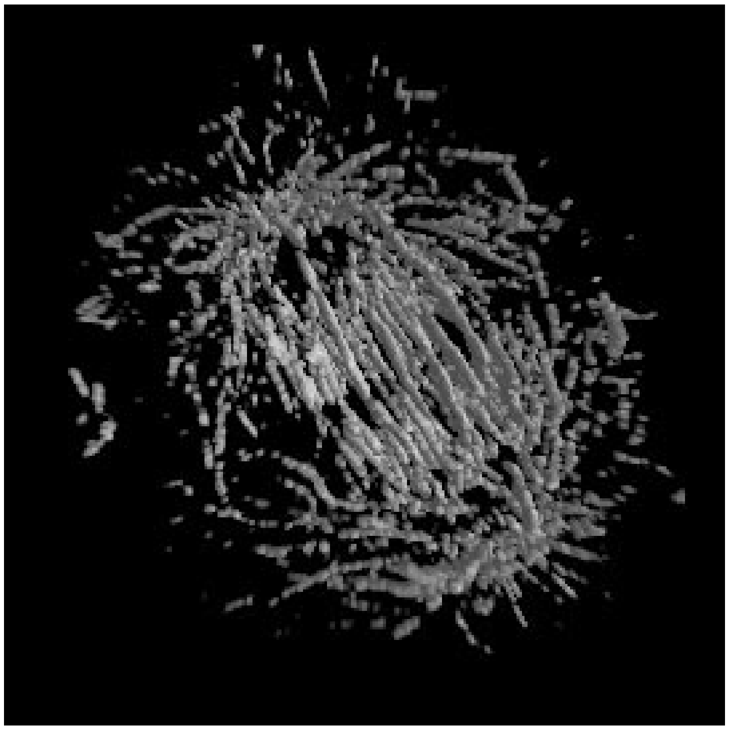


FIGURE 9. Animated sequence of the dataset in Figure 8. (a) Sequence from aliased animation. (b) Sequence from anti-aliased animation.


(a)

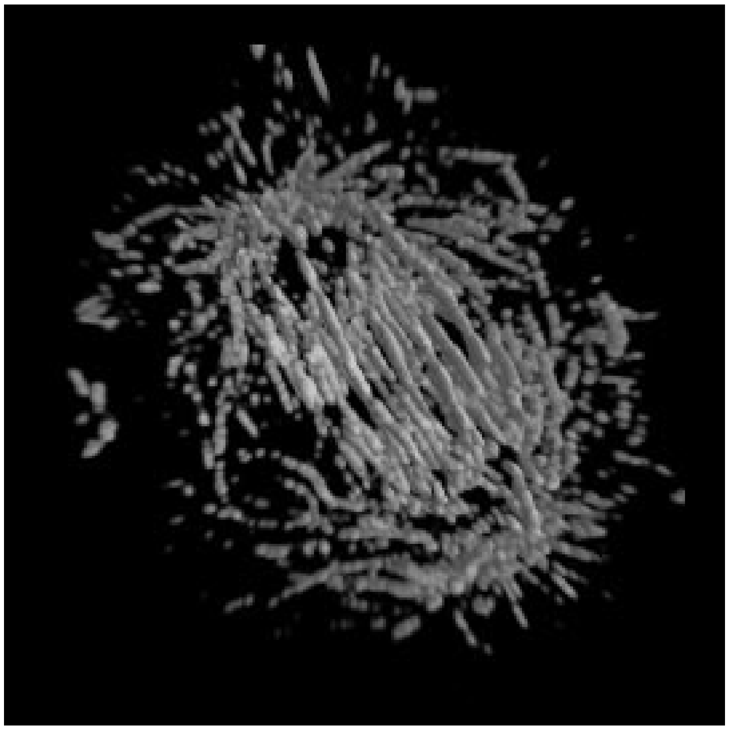

(b)

FIGURE 10. Rendered image of a microtubule dataset acquired from confocal microscopy. (a) Rendered with standard splatting. (b) Rendered with anti-aliased splatting.

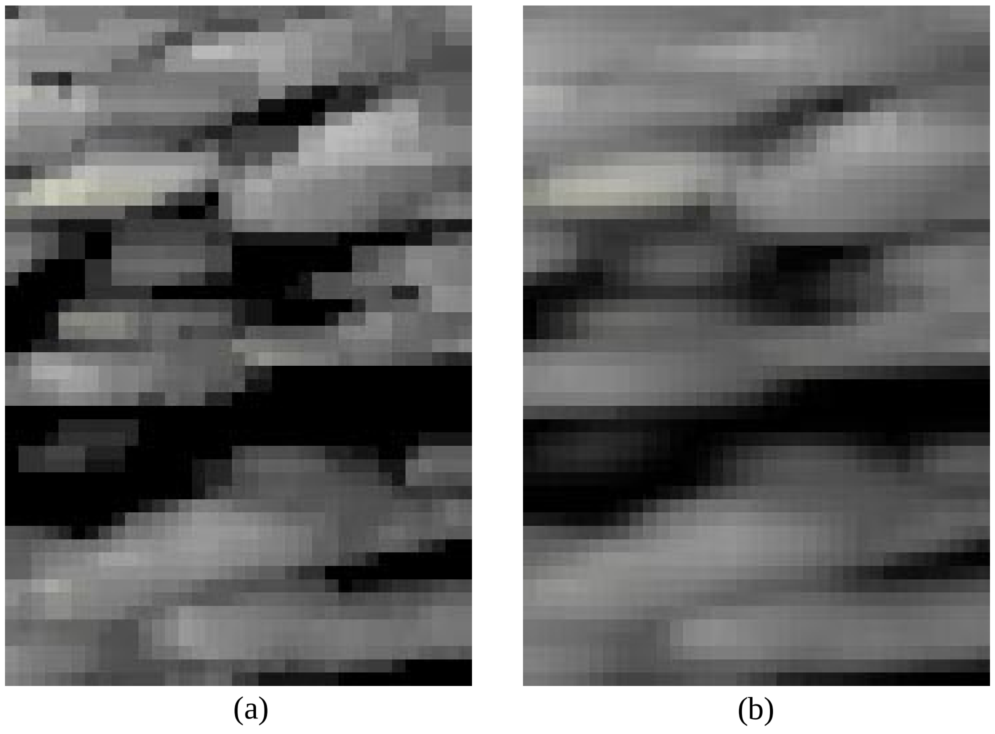


FIGURE 11. Animated sequence of the dataset in Figure 10. (a) Sequence from aliased animation. (b) Sequence from anti-aliased animation.

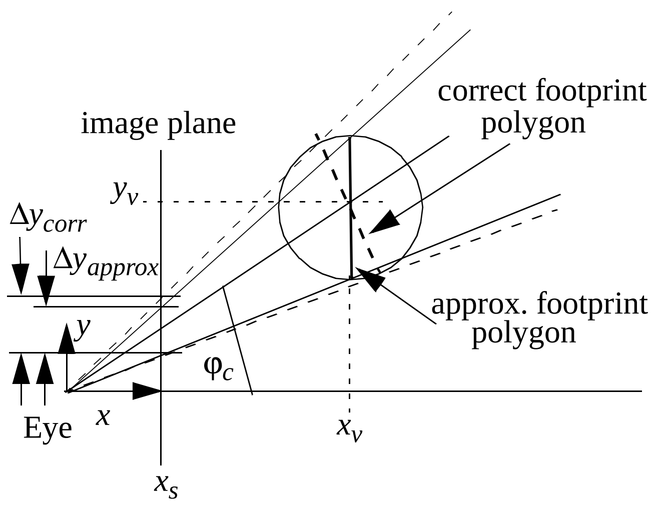


FIGURE 12. Errors from the non-perpendicular alignment of the footprint polygon.

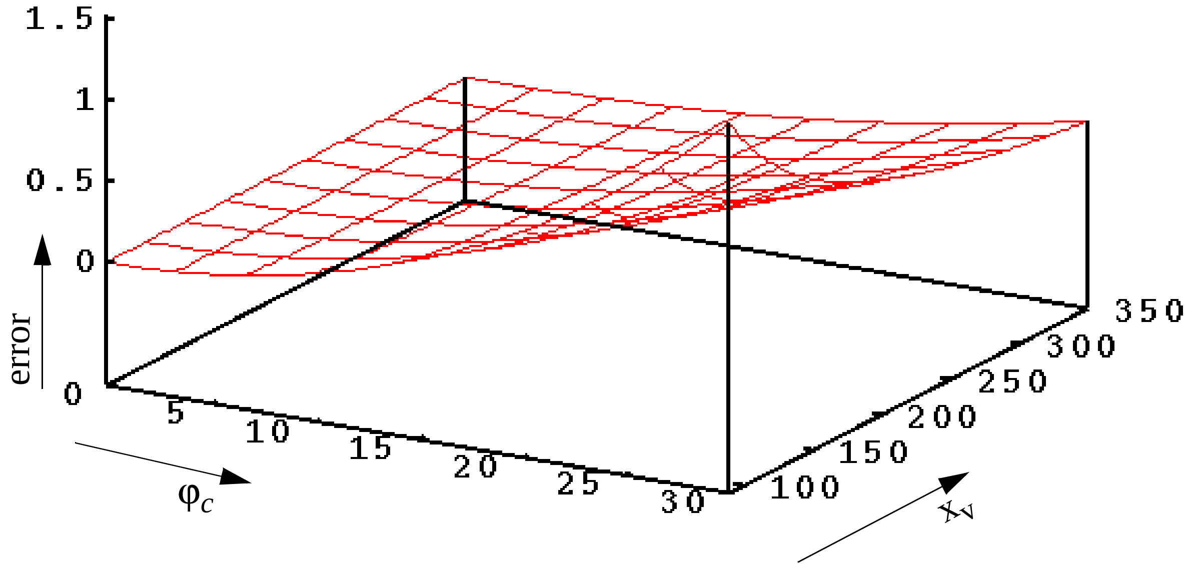


FIGURE 13. Error behavior of the non-perpendicular alignment of the footprint polygon, plotted as a function of x_v (distance from the view plane) and φ_c (viewing frustum half-angle). The maximum error is 1.46.

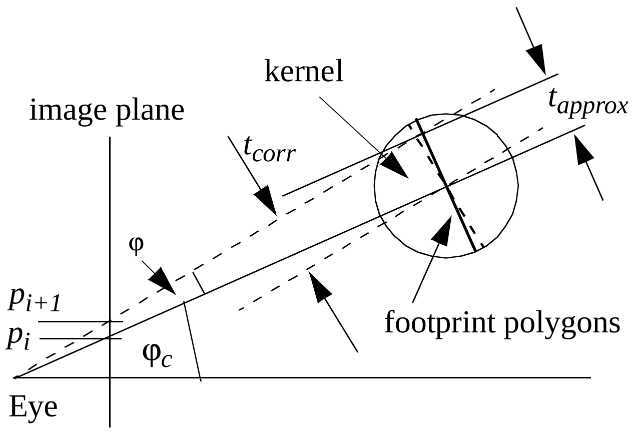


FIGURE 14. Errors from the non-perpendicular traversal of the footprint polygon.

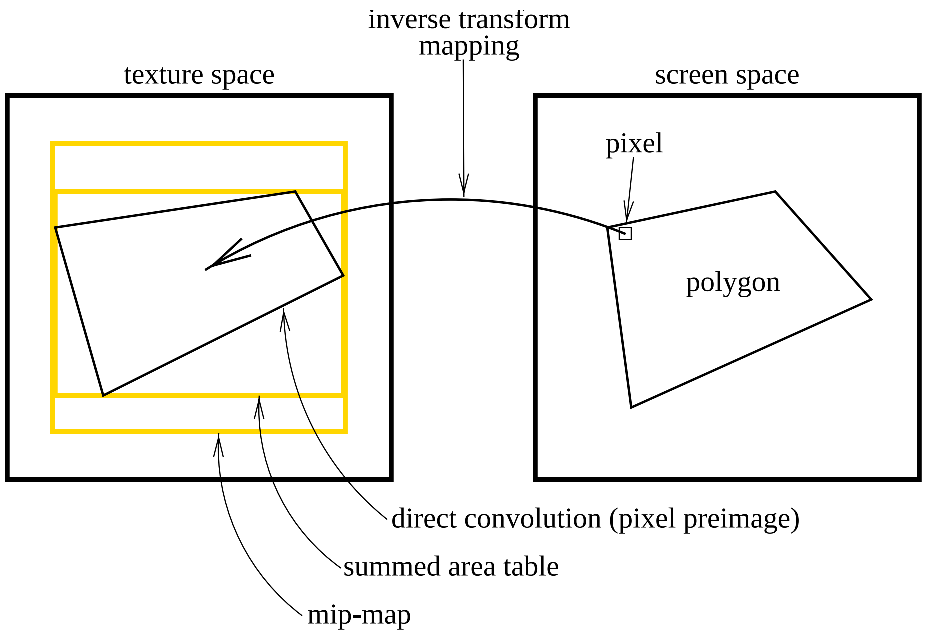


FIGURE 15. Traditional texture mapping, showing the pixel preimage for direct convolution, a summed area table, and mip-mapping.

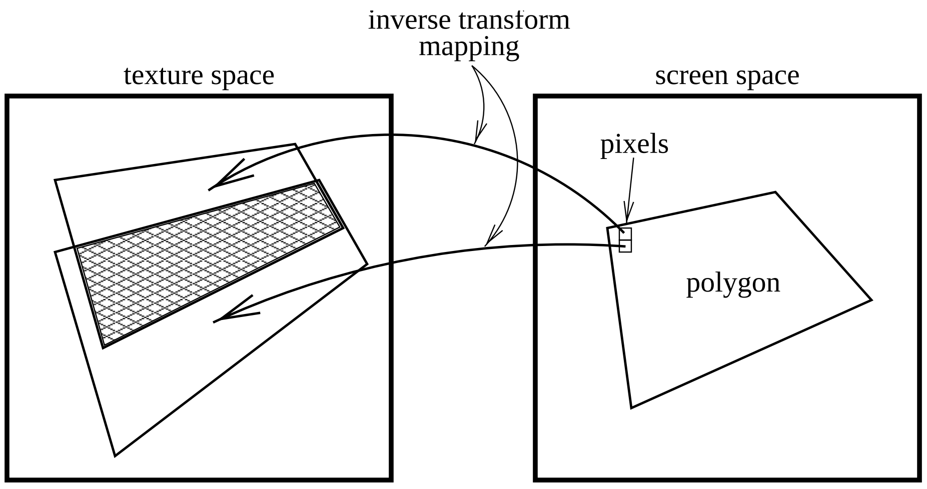


FIGURE 16. Traditional texture mapping, showing the preimages of two adjacent pixels.

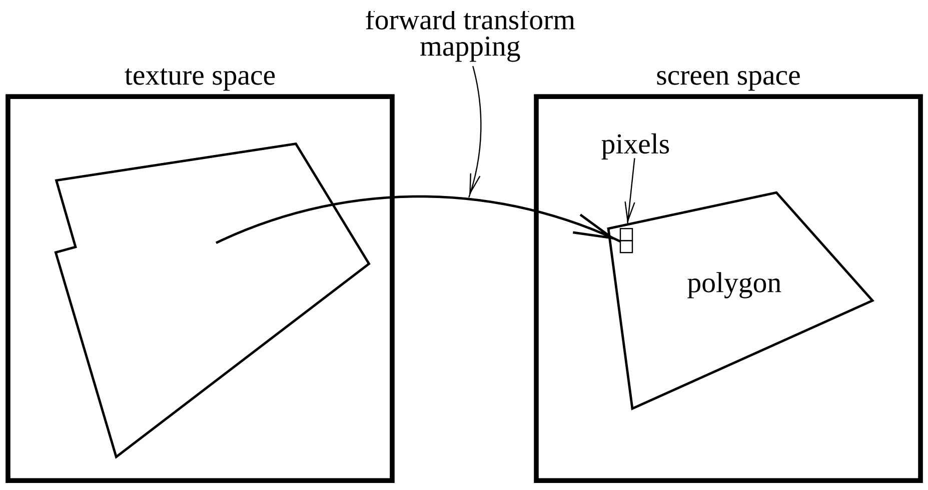


FIGURE 17. Texture mapping with the new method discussed in this paper.

Author Affiliations

- J. Edward Swan II is with The Naval Research Laboratory, Code 5580, 4555 Overlook Ave SW, Washington, DC 20375-5320. E-Mail: swan@acm.org
- Klaus Mueller is with the Department of Computer and Information Science, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210-1277. E-Mail: mueller@cis.ohio-state.edu
- Torsten Möller is with the Department of Computer and Information Science and the Advanced Computing Center for the Arts and Design, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210-1277. E-Mail: moeller@cis.ohio-state.edu
- Naeem Shareef is with the Department of Computer and Information Science and the Advanced Computing Center for the Arts and Design, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210-1277. E-Mail: shareef@cis.ohio-state.edu
- Roger Crawfis is with the Department of Computer and Information Science and the Advanced Computing Center for the Arts and Design, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210-1277. E-Mail: crawfis@cis.ohio-state.edu
- Roni Yagel is with the Department of Computer and Information Science and the Advanced Computing Center for the Arts and Design, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210-1277. E-Mail: yagel@cis.ohio-state.edu

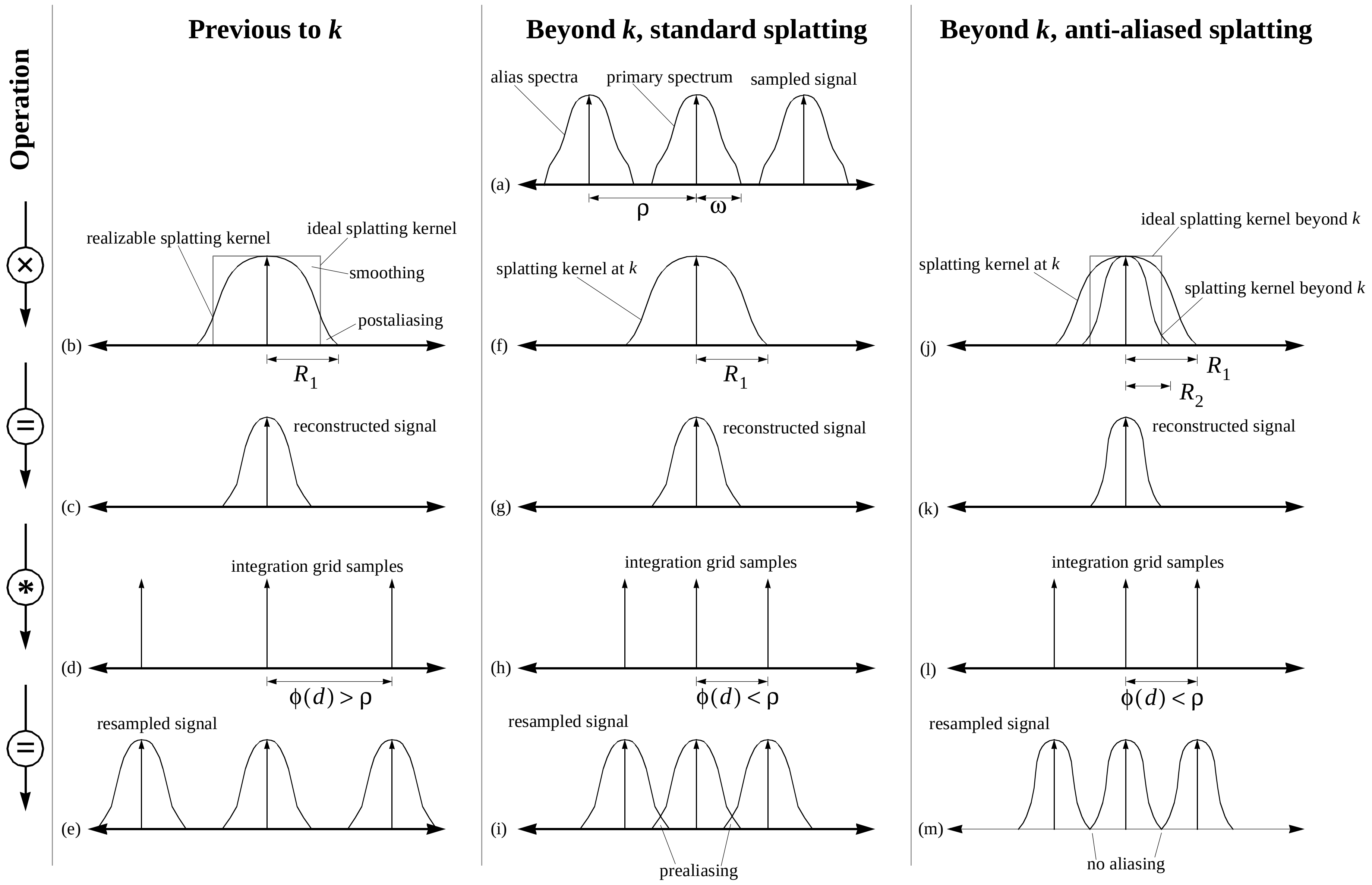


FIGURE 18. The behavior of the anti-aliased splatting technique in the frequency domain.

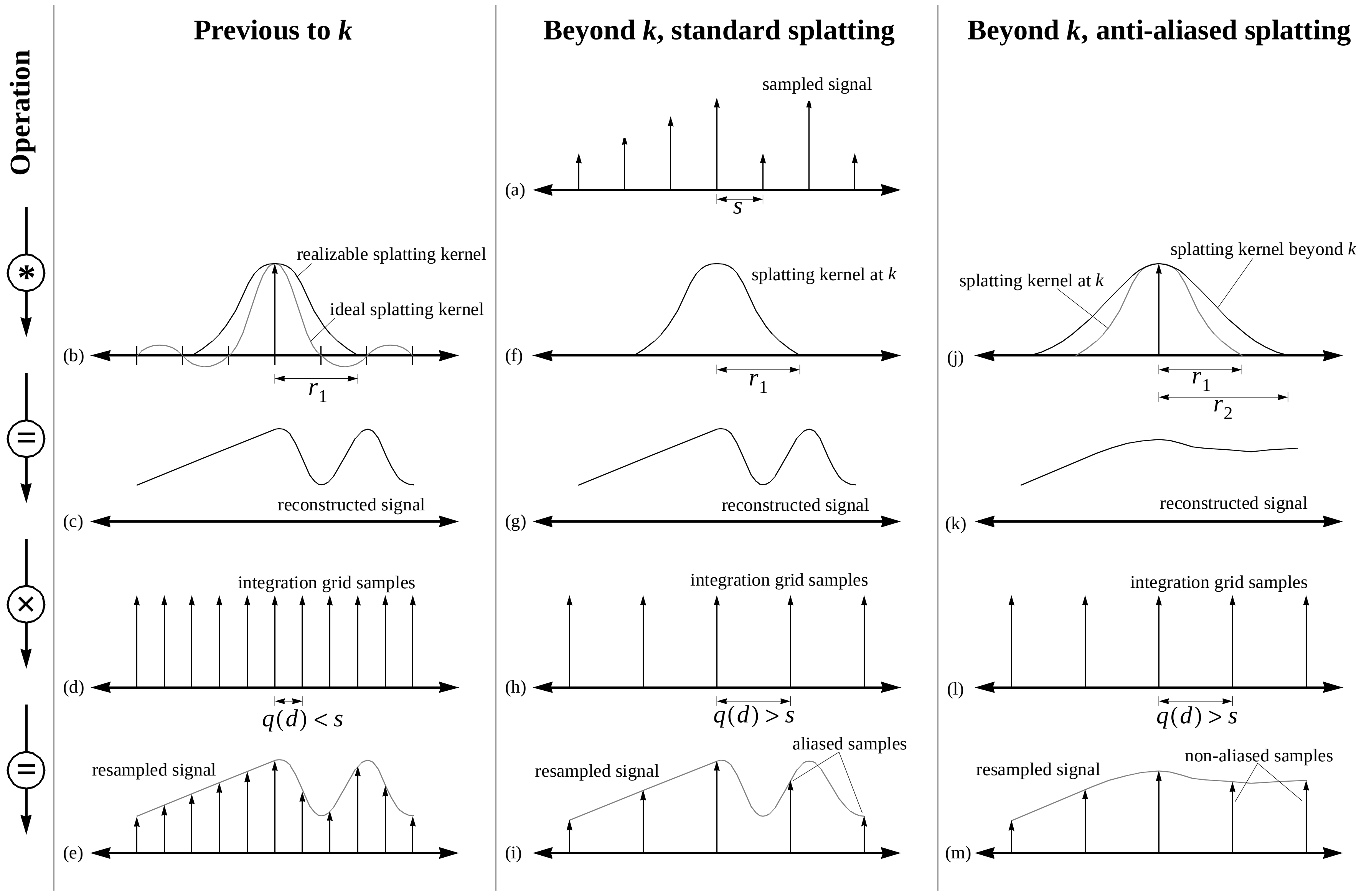


FIGURE 19. The behavior of the anti-aliased splatting technique in the spatial domain.