
DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU


HARVARD LIBRARY
Office for Scholarly Communication

EWA Splatting

Citation
Zwicker, Matthias, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. 2002. EWA Splatting. IEEE Transactions on Visualization and Computer Graphics 8(3): 223-238.

Published Version
doi:10.1109/TVCG.2002.1021576

Permanent link
http://nrs.harvard.edu/urn-3:HUL.InstRepos:4138240

Terms of Use
This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

To appear in the Proceedings of IEEE Visualization 2001

EWA Volume Splatting

Matthias Zwicker * Hanspeter Pfister † Jeroen van Baar† Markus Gross*

Abstract

In this paper we present a novel framework for direct volume rendering using a splatting approach based on elliptical Gaussian kernels. To avoid aliasing artifacts, we introduce the concept of a resampling filter combining a reconstruction with a low-pass kernel. Because of the similarity to Heckbert's EWA (elliptical weighted average) filter for texture mapping we call our technique EWA volume splatting. It provides high image quality without aliasing artifacts or excessive blurring even with non-spherical kernels. Hence it is suitable for regular, rectilinear, and irregular volume data sets. Moreover, our framework introduces a novel approach to compute the footprint function. It facilitates efficient perspective projection of arbitrary elliptical kernels at very little additional cost. Finally, we show that EWA volume reconstruction kernels can be reduced to surface reconstruction kernels. This makes our splat primitive universal in reconstructing surface and volume data.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Keywords: Volume Rendering, Splatting, Antialiasing.

1 Introduction

Volume rendering is an important technique in visualizing acquired and simulated data sets in scientific and engineering applications. The ideal volume rendering algorithm reconstructs a continuous function in 3D, transforms this 3D function into screen space, and then evaluates opacity integrals along line-of-sights. In 1989, Westover [18, 19] introduced *splatting* for interactive volume rendering, which approximates this procedure. Splatting algorithms interpret volume data as a set of particles that are absorbing and emitting light. Line integrals are precomputed across each particle separately, resulting in *footprint functions*. Each footprint spreads its contribution in the image plane. These contributions are composited back to front into the final image.

We introduce a new footprint function for volume splatting algorithms integrating an elliptical Gaussian reconstruction kernel and a low-pass filter. Our derivation proceeds along similar lines as Heckbert's *elliptical weighted average* (EWA) texture filter [4], therefore we call our algorithm *EWA volume splatting*.

EWA volume rendering is attractive because it prevents aliasing artifacts in the output image while avoiding excessive blurring. Moreover, it works with arbitrary elliptical Gaussian reconstruction kernels and efficiently supports perspective projection. Our method is based on a novel framework to compute the footprint function, which relies on the transformation of the volume data to so-called *ray space*. This transformation is equivalent to perspective projection. By using its local affine approximation at each voxel, we derive an analytic expression for the EWA footprint in screen space. The rasterization of the footprint is performed using forward differencing requiring only one 1D footprint table for all reconstruction kernels and any viewing direction.

Our splat primitive can be integrated easily into conventional splatting algorithms. Because of its flexibility, it can be utilized to render rectilinear, curvilinear, or unstructured volume data sets. By flattening the 3D Gaussian kernel along the volume gradient we will show that EWA volume splats reduce to surface splats that are suitable for high quality iso-surface rendering.

The paper is organized as follows: We discuss previous work in Section 2. Next, we review a typical volume rendering pipeline and the volume rendering equation in Section 3. Specifically, we elaborate how the volume rendering equation is computed by typical splatting algorithms. In Section 4, we present our EWA volume rendering framework. We start by analyzing the aliasing problem due to improper sampling of the output function resulting from volume rendering. In a next step, we introduce the EWA resampling filter, which integrates an arbitrary elliptical Gaussian reconstruction kernel and a Gaussian low-pass filter. Our derivation is based on the local affine transformation of the volume data such that the reconstruction kernels can be integrated analytically. Furthermore, we show how the EWA reconstruction kernels can be continuously adapted from volumes to surfaces in Section 5. Finally, Sections 6 and 7 discuss our implementation and results.

2 Previous Work

The original work on splatting was presented by Westover [18]. Basic splatting algorithms suffer from inaccurate visibility determination when compositing the splats from back to front. This leads to visible artifacts such as color bleeding. Later, Westover [19] solved the problem using an axis-aligned sheet buffer. However, this technique is plagued by disturbing popping artifacts in animations. Recently, Mueller and Crawfis [14] proposed to align the sheet buffers parallel to the image plane instead of parallel to an axis of the volume data. Additionally, they splat several slices of each reconstruction kernel separately. This technique is similar to *slice-based* volume rendering [2, 1] and does not suffer from popping artifacts. Mueller and Yagel [15] combine splatting with ray casting techniques to accelerate rendering with perspective projection. Laur and Hanrahan [7] describe a hierarchical splatting algorithm enabling progressive refinement during rendering. Furthermore, Lippert [9] introduced a splatting algorithm that directly uses a wavelet representation of the volume data.

Westover's original framework does not deal with sampling rate changes due to perspective projections. Aliasing artifacts may occur in areas of the volume where the sampling rate of diverging rays falls below the volume grid sampling rate. Swan et al. [17] use a distance-dependent stretch of the footprints to make them act as low-pass filters. This antialiasing method is closely related to EWA volume splatting, and we will discuss it further in Section 7.

Additional care has to be taken if the 3D kernels are not radially symmetric, as is the case for rectilinear, curvilinear, or irregular grids. In addition, for an arbitrary position in 3D, the contributions from all kernels must sum up to one. Otherwise, artifacts such as splotches occur in the image. For rectilinear grids, Westover [19] proposes using elliptical footprints that are warped back to a circular footprint. To render curvilinear grids, Mao et al. [10] use

*ETH Zürich, Switzerland. Email: [zwicker,grossm]@inf.ethz.ch
†MERL, Cambridge, MA. Email: [pfister,jeroen]@merl.com

stochastic Poisson resampling to generate a set of new points whose kernels are spheres or ellipsoids. They compute the elliptical footprints very similar to Westover [19]. As pointed out in Section 4, our technique can be used with irregular grids to efficiently and accurately project and rasterize the elliptical splat kernels.

We develop EWA volume splatting along similar lines to the seminal work of Heckbert [4], who introduced EWA filtering to avoid aliasing of surface textures. We recently extended his framework to represent and render texture functions on irregularly point-sampled surfaces [21]. Section 5 will show the connection between EWA volume and surface splatting.

3 Preliminaries

3.1 The Volume Rendering Pipeline

We distinguish two fundamental approaches to volume rendering: backward mapping algorithms that shoot rays through pixels on the image plane into the volume data, and forward mapping algorithms that map the data onto the image plane. In the following discussion, we will describe a forward mapping technique. Mapping the data onto the image plane involves a sequence of intermediate steps where the data is transformed to different coordinate systems, as in conventional rendering pipelines. We introduce our terminology in Figure 1. Note that the terms *space* and *coordinate system* are synonymous. The figure summarizes a *forward mapping volume rendering pipeline*, where the data flows from the left to the right.

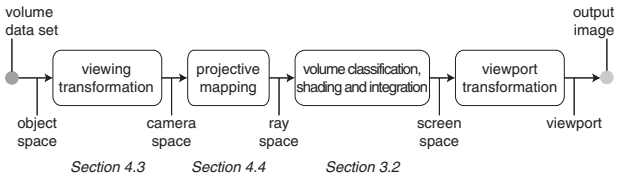


Figure 1: *The forward mapping volume rendering pipeline.*

As an overview, we briefly describe the coordinate systems and transformations that are relevant for our technique. The volume data is initially given in *object coordinates*. To render the data from an arbitrary viewpoint, it is first mapped to *camera space* using the viewing transformation. We deal with the effect of this transformation in Section 4.3. The camera coordinate system is defined such that its origin is at the center of projection.

We further transform the data to *ray space*, which is introduced in Section 3.2. Ray space is a non-cartesian coordinate system that enables an easy formulation of the volume rendering equation. In ray space, the viewing rays are parallel to a coordinate axis, facilitating analytical integration of the volume function. We present the transformation from camera to ray space in Section 4.4; it is a key element of our technique. Since its purpose is similar to the projective transform used in rendering pipelines such as OpenGL, it is also called the *projective mapping*.

Evaluating the volume rendering equation results in a 2D image in *screen space*. In a final step, this image is transformed to *viewport coordinates*. Focusing on the essential aspects of our technique, we are not covering the viewport transformation in the following explanations. However, it can be easily incorporated in an implementation. Moreover, we do not discuss volume classification and shading in a forward mapping pipeline, but refer to [13] or [20] for a thorough discussion.

3.2 Splatting Algorithms

We review the low albedo approximation of the volume rendering equation [5, 12] as used for fast, direct volume rendering [19, 6, 13, 8]. The left part of Figure 2 illustrates the corresponding situation in 2D. Starting from this form of the rendering equation, we discuss several simplifying assumptions leading to the well known *splatting* formulation. Because of their efficiency, splatting algorithms [19, 13] belong to the most popular forward mapping volume rendering techniques.

We slightly modify the conventional notation, introducing our concept of ray space. We denote a point in ray space by a column vector of three coordinates $\mathbf{x} = (x_0, x_1, x_2)^T$. Given a center of projection and a projection plane, these three coordinates are interpreted geometrically as follows: The coordinates x_0 and x_1 specify a point on the projection plane. The ray intersecting the center of projection and the point (x_0, x_1) on the projection plane is called a viewing ray. Using the abbreviation $\hat{\mathbf{x}} = (x_0, x_1)^T$, we refer to the viewing ray passing through (x_0, x_1) as $\hat{\mathbf{x}}$. The third coordinate x_2 specifies the Euclidean distance from the center of projection to a point on the viewing ray. To simplify the notation, we will use any of the synonyms $\mathbf{x}$, $(\hat{\mathbf{x}}, x_2)^T$, or $(x_0, x_1, x_2)^T$ to denote a point in ray space.

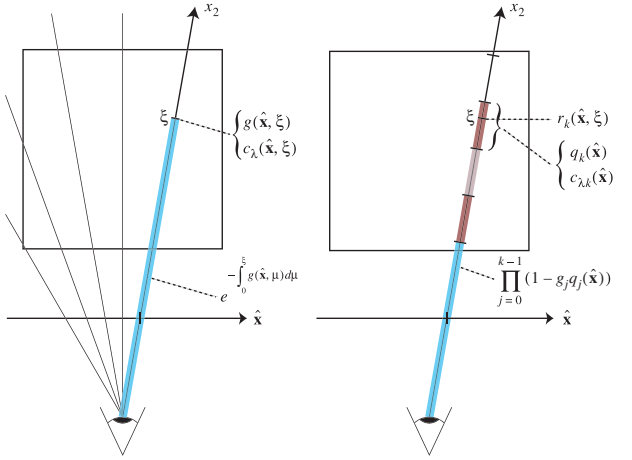


Figure 2: *Volume rendering. Left: Illustrating the volume rendering equation in 2D. Right: Approximations in typical splatting algorithms.*

The volume rendering equation describes the light intensity $I_\lambda(\hat{\mathbf{x}})$ at wavelength λ that reaches the center of projection along the ray $\hat{\mathbf{x}}$ with length L:

$$I_\lambda(\hat{\mathbf{x}}) = \int_0^L c_\lambda(\hat{\mathbf{x}}, \xi) g(\hat{\mathbf{x}}, \xi) e^{-\int_0^\xi g(\hat{\mathbf{x}}, \mu)\, d\mu}\, d\xi, \qquad (1)$$

where $g(\mathbf{x})$ is the *extinction function* that defines the rate of light occlusion, and $c_\lambda(\mathbf{x})$ is an emission coefficient. The exponential term can be interpreted as an *attenuation factor*. Finally, the product $c_\lambda(\mathbf{x})g(\mathbf{x})$ is also called the *source term* [12], describing the light intensity scattered in the direction of the ray $\hat{\mathbf{x}}$ at the point x_2.

Now we assume that the extinction function is given as a weighted sum of coefficients g_k and reconstruction kernels $r_k(\mathbf{x})$. This corresponds to a physical model where the volume consists of individual particles that absorb and emit light. Hence the extinction function is:

$$g(\mathbf{x}) = \sum_k g_k r_k(\mathbf{x}). \qquad (2)$$

In this mathematical model, the reconstruction kernels $r_k(\mathbf{x})$ reflect position and shape of individual particles. The particles can be irregularly spaced and may differ in shape, hence the representation in (2) is not restricted to regular data sets. We substitute (2) into (1), yielding:

$$I_\lambda(\hat{\mathbf{x}}) = \sum_k \left(\int_0^L c_\lambda(\hat{\mathbf{x}}, \xi) g_k r_k(\hat{\mathbf{x}}, \xi) \prod_j e^{-g_j \int_0^\xi r_j(\hat{\mathbf{x}}, \mu)\, d\mu}\, d\xi \right). \qquad (3)$$

To compute this function numerically, splatting algorithms make several simplifying assumptions, illustrated in the right part of Figure 2. Usually the reconstruction kernels $r_k(\mathbf{x})$ have local support. The splatting approach assumes that these local support areas do not overlap along a ray $\hat{\mathbf{x}}$, and the reconstruction kernels are ordered front to back. We also assume that the emission coefficient is constant in the support of each reconstruction kernel along a ray, hence we use the notation $c_{\lambda k}(\hat{\mathbf{x}}) = c_\lambda(\hat{\mathbf{x}}, x_2)$, where $(\hat{\mathbf{x}}, x_2)$ is in the support of r_k. Moreover, we approximate the exponential function with the first two terms of its Taylor expansion, thus $e^x \approx 1-x$. Finally, we ignore self-occlusion. Exploiting these assumptions, we rewrite (3), yielding:

$$I_\lambda(\hat{\mathbf{x}}) = \sum_k c_{\lambda k}(\hat{\mathbf{x}}) g_k q_k(\hat{\mathbf{x}}) \prod_{j=0}^{k-1} (1 - g_j q_j(\hat{\mathbf{x}})), \tag{4}$$

where $q_k(\hat{\mathbf{x}})$ denotes an integrated reconstruction kernel, hence:

$$q_k(\hat{\mathbf{x}}) = \int_{\mathbb{R}} r_k(\hat{\mathbf{x}}, x_2)\, dx_2. \tag{5}$$

Equation (4) is the basis for all splatting algorithms. Westover [19] introduced the term *footprint function* for the integrated reconstruction kernels q_k. The footprint function is a 2D function that specifies the contribution of a 3D kernel to each point on the image plane. Integrating a volume along a viewing ray is analogous to projecting a point on a surface onto the image plane, hence the coordinates $\hat{\mathbf{x}} = (x_0, x_1)^T$ are also called *screen coordinates*, and we say that $I_\lambda(\hat{\mathbf{x}})$ and $q_k(\hat{\mathbf{x}})$ are defined in *screen space*.

Splatting is attractive because of its efficiency, which it derives from the use of pre-integrated reconstruction kernels. Therefore, during volume integration each sample point along a viewing ray is computed using a 2D convolution. In contrast, ray-casting methods require a 3D convolution for each sample point. This provides splatting algorithms with an inherent advantage in rendering efficiency. Moreover, splatting facilitates the use of higher quality kernels with a larger extent than the trilinear kernels typically employed by ray-casting. On the other hand, basic splatting methods are plagued by artifacts because of incorrect visibility determination. This problem is unavoidably introduced by the assumption that the reconstruction kernels do not overlap and are ordered back to front. It has been successfully addressed by several authors as mentioned in Section 2. In contrast, our main contribution is a novel splat primitive that provides high quality antialiasing and efficiently supports elliptical kernels. We believe that our novel primitive is compatible with all state-of-the-art algorithms.

4 The EWA Volume Resampling Filter

4.1 Aliasing in Volume Splatting

Aliasing is a fundamental problem of any rendering algorithm, arising whenever a rendered image or a part of it is sampled to a discrete raster grid, i.e., the pixel grid. Aliasing leads to visual artifacts such as jagged silhouette edges and *Moiré* patterns in textures. Typically, these problems become most disturbing during animations. From a signal processing point of view, aliasing is well understood: before a continuous function is sampled to a regular sampling grid, it has to be band-limited to respect the Nyquist frequency of the grid. This guarantees that there are no aliasing artifacts in the sampled image. In this section we provide a systematic analysis on how to band-limit the splatting equation.

The splatting equation (4) represents the output image as a *continuous* function $I_\lambda(\hat{\mathbf{x}})$ in screen space. In order to properly sample this function to a *discrete* output image without aliasing artifacts, it has to be band-limited to match the Nyquist frequency of the discrete image. In theory, we achieve this band-limitation by convolving $I_\lambda(\hat{\mathbf{x}})$ with an appropriate low-pass filter $h(\hat{\mathbf{x}})$, yielding the *antialiased* splatting equation

$$(I_\lambda \otimes h)(\hat{\mathbf{x}}) = \int_{\mathbb{R}^2} \sum_k c_{\lambda k}(\eta) g_k q_k(\eta) \prod_{j=0}^{k-1} (1 - g_j q_j(\eta))\, h(\hat{\mathbf{x}} - \eta)\, d\eta. \tag{6}$$

Although $I_\lambda(\hat{\mathbf{x}})$ is formulated as a continuous function in (4), in practice this function is evaluated only at discrete positions, i.e., the pixel centers. Therefore we cannot evaluate (6), which requires that $I_\lambda(\hat{\mathbf{x}})$ is available as a continuous function.

However, we make two simplifying assumptions to rearrange the integral in (6). This leads to an approximation that can be evaluated efficiently. First, we assume that the emission coefficient is approximately constant in the support of each footprint function q_k, hence $c_{\lambda k}(\hat{\mathbf{x}}) \approx c_{\lambda k}$ for all $\hat{\mathbf{x}}$ in the support area. Together with the assumption that the emission coefficient is constant in the support of each reconstruction kernel along a *viewing ray*, this means that the emission coefficient is constant in the complete *3D support* of each reconstruction kernel. In other words, we ignore the effect of shading for antialiasing. Note that this is the common approach for antialiasing surface textures as well.

Additionally, we assume that the attenuation factor has an approximately constant value o_k in the support of each footprint function. Hence:

$$\prod_{j=0}^{k-1} (1 - g_j q_j(\hat{\mathbf{x}})) \approx o_k \tag{7}$$

for all $\hat{\mathbf{x}}$ in the support area. A variation of the attenuation factor indicates that the footprint function is partially covered by a more opaque region in the volume data. Therefore this variation can be interpreted as a "soft" edge. Ignoring such situations means that we cannot prevent edge aliasing. Again, this is similar to rendering surfaces, where edge and texture aliasing are handled by different algorithms as well.

Exploiting these simplifications, we can rewrite (6) to:

$$\begin{aligned}(I_\lambda \otimes h)(\hat{\mathbf{x}}) &\approx \sum_k c_{\lambda k} o_k g_k \int_{\mathbb{R}^2} q_k(\eta) h(\hat{\mathbf{x}} - \eta)\, d\eta \\ &= \sum_k c_{\lambda k} o_k g_k (q_k \otimes h)(\hat{\mathbf{x}}).\end{aligned}$$

Following Heckbert's terminology [4], we call:

$$\rho_k(\hat{\mathbf{x}}) = (q_k \otimes h)(\hat{\mathbf{x}}) \tag{8}$$

an *ideal resampling filter*, combining a footprint function q_k and a low-pass kernel h. Hence, we can approximate the antialiased splatting equation (6) by replacing the footprint function q_k in the original splatting equation (4) with the resampling filter ρ_k. This means that instead of band-limiting the output function $I_\lambda(\hat{\mathbf{x}})$ directly, we band-limit each footprint function separately. Under the assumptions described above, we get a splatting algorithm that produces a band-limited output function respecting the Nyquist frequency of the raster image, therefore avoiding aliasing artifacts. Remember that the reconstruction kernels are integrated in ray space, resulting in footprint functions that vary significantly in size and shape across the volume. Hence the resampling filter in (8) is strongly *space variant*.

Swan et al. presented an antialiasing technique for splatting [17] that is based on a uniform scaling of the reconstruction kernels to band-limit the extinction function. Their technique produces similar results as our method for radially symmetric kernels. However, for more general kernels, e.g., elliptical kernels, uniform scaling

is a poor approximation of ideal low-pass filtering. Aliasing artifacts cannot be avoided without introducing additional blurriness. On the other hand, our method provides non-uniform scaling in these cases, leading to superior image quality as illustrated in Section 7. Moreover, our analysis above shows that band-limiting the extinction function does not guarantee aliasing free images. Because of shading and edges, frequencies above the Nyquist limit persist. However, these effects are not discussed in [17].

4.2 Elliptical Gaussian Kernels

We choose elliptical Gaussians as reconstruction kernels and low-pass filters, since they provide certain features that are crucial for our technique: Gaussians are closed under affine mappings and convolution, and integrating a 3D Gaussian along one coordinate axis results in a 2D Gaussian. These properties enable us to analytically compute the resampling filter in (8) as a single 2D Gaussian, as will be shown below. In this section, we summarize the mathematical features of the Gaussians that are exploited in our derivation in the following sections. More details on Gaussians can be found in Heckbert's master's thesis [4].

We define an elliptical Gaussian $\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p})$ centered at a point $\mathbf{p}$ with a variance matrix $\mathbf{V}$ as:

$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}, \tag{9}$$

where $|\mathbf{V}|$ is the determinant of $\mathbf{V}$. In this form, the Gaussian is normalized to a unit integral. In the case of volume reconstruction kernels, $\mathcal{G}_{\mathbf{V}}$ is a 3D function, hence $\mathbf{V}$ is a symmetric 3×3 matrix and $\mathbf{x}$ and $\mathbf{p}$ are column vectors $(x_0,x_1,x_2)^T$ and $(p_0,p_1,p_2)^T$, respectively. We can easily apply an arbitrary affine mapping $\mathbf{u} = \Phi(\mathbf{x})$ to this Gaussian. Let us define the affine mapping as $\Phi(\mathbf{x}) = \mathbf{M}\mathbf{x}+\mathbf{c}$, where $\mathbf{M}$ is a 3×3 matrix and $\mathbf{c}$ is a vector $(c_0,c_1,c_2)^T$. We substitute $\mathbf{x} = \Phi^{-1}(\mathbf{u})$ in (9), yielding:

$$\mathcal{G}_{\mathbf{V}}(\Phi^{-1}(\mathbf{u})-\mathbf{p}) = \frac{1}{|\mathbf{M}^{-1}|}\mathcal{G}_{\mathbf{M}\mathbf{V}\mathbf{M}^T}(\mathbf{u}-\Phi(\mathbf{p})). \tag{10}$$

Moreover, convolving two Gaussians with variance matrices $\mathbf{V}$ and $\mathbf{Y}$ results in another Gaussian with variance matrix $\mathbf{V}+\mathbf{Y}$:

$$(\mathcal{G}_{\mathbf{V}} \otimes \mathcal{G}_{\mathbf{Y}})(\mathbf{x}-\mathbf{p}) = \mathcal{G}_{\mathbf{V}+\mathbf{Y}}(\mathbf{x}-\mathbf{p}). \tag{11}$$

Finally, integrating a 3D Gaussian $\mathcal{G}_{\mathbf{V}}$ along one coordinate axis yields a 2D Gaussian $\mathcal{G}_{\hat{\mathbf{V}}}$, hence:

$$\int_{\mathbb{R}} \mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p})\,dx_2 = \mathcal{G}_{\hat{\mathbf{V}}}(\hat{\mathbf{x}}-\hat{\mathbf{p}}), \tag{12}$$

where $\hat{\mathbf{x}} = (x_0,x_1)^T$ and $\hat{\mathbf{p}} = (p_0,p_1)^T$. The 2×2 variance matrix $\hat{\mathbf{V}}$ is easily obtained from the 3×3 matrix $\mathbf{V}$ by skipping the third row and column:

$$\mathbf{V} = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \Leftrightarrow \begin{pmatrix} a & b \\ b & d \end{pmatrix} = \hat{\mathbf{V}}. \tag{13}$$

In the following sections, we describe how to map arbitrary elliptical Gaussian reconstruction kernels from object to ray space. Our derivation results in an analytic expression for the kernels in ray space $r_k(\mathbf{x})$ as in Equation (2). We will then be able to analytically integrate the kernels according to Equation (5) and to convolve the footprint function q_k with a Gaussian low-pass filter h as in Equation (8), yielding an elliptical Gaussian resampling filter ρ_k.

4.3 The Viewing Transformation

The reconstruction kernels are initially given in *object space*, which has coordinates $\mathbf{t} = (t_0,t_1,t_2)^T$. Let us denote the Gaussian reconstruction kernels in object space by $r''_k(\mathbf{t}) = \mathcal{G}_{\mathbf{V}''_k}(\mathbf{t}-\mathbf{t}_k)$, where $\mathbf{t}_k$ are the voxel positions in object space. For regular volume data sets, the variance matrices $\mathbf{V}''_k$ are usually identity matrices. For rectilinear data sets, they are diagonal matrices where the matrix elements contain the squared distances between voxels along each coordinate axis. Curvilinear and irregular grids have to be resampled to a more regular structure in general. For example, Mao et al. [11] describe a stochastic sampling approach with a method to compute the variance matrices for curvilinear volumes.

We denote camera coordinates by a vector $\mathbf{u} = (u_0,u_1,u_2)^T$. Object coordinates are transformed to camera coordinates using an affine mapping $\mathbf{u} = \varphi(\mathbf{t})$, called *viewing transformation*. It is defined by a matrix $\mathbf{W}$ and a translation vector $\mathbf{d}$ as $\varphi(\mathbf{t}) = \mathbf{W}\mathbf{t}+\mathbf{d}$. We transform the reconstruction kernels $\mathcal{G}_{\mathbf{V}''_k}(\mathbf{t}-\mathbf{t}_k)$ to camera space by substituting $\mathbf{t} = \varphi^{-1}(\mathbf{u})$ and using Equation (10):

$$\mathcal{G}_{\mathbf{V}''_k}(\varphi^{-1}(\mathbf{u})-\mathbf{t}_k) = \frac{1}{|\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}'_k}(\mathbf{u}-\mathbf{u}_k) = r'_k(\mathbf{u}), \tag{14}$$

where $\mathbf{u}_k = \varphi(\mathbf{t}_k)$ is the center of the Gaussian in camera coordinates and $r'_k(\mathbf{u})$ denotes the reconstruction kernel in camera space. According to (10), the variance matrix in camera coordinates $\mathbf{V}'_k$ is given by $\mathbf{V}'_k = \mathbf{W}\mathbf{V}''_k\mathbf{W}^T$.

4.4 The Projective Transformation

The projective transformation converts camera coordinates to ray coordinates as illustrated in Figure 3. Camera space is defined such that the origin of the camera coordinate system is at the center of projection and the projection plane is the plane $u_2 = 1$. Camera space and ray space are related by the mapping $\mathbf{x} = \mathbf{m}(\mathbf{u})$. Using the definition of ray space from Section 3, $\mathbf{m}(\mathbf{u})$ and its inverse $\mathbf{m}^{-1}(\mathbf{x})$ are therefore given by:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \mathbf{m}(\mathbf{u}) = \begin{pmatrix} u_0/u_2 \\ u_1/u_2 \\ \|(u_0,u_1,u_2)^T\| \end{pmatrix} \tag{15}$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l\cdot x_2 \\ x_1/l\cdot x_2 \\ 1/l\cdot x_2 \end{pmatrix}, \tag{16}$$

where $l = \|(x_0,x_1,1)^T\|$.

Unfortunately, these mappings are not affine, so we cannot apply Equation (10) directly to transform the reconstruction kernels from camera to ray space. To solve this problem, we introduce the *local affine approximation* $\mathbf{m}_{\mathbf{u}_k}$ of the projective transformation. It is defined by the first two terms of the Taylor expansion of $\mathbf{m}$ at the point $\mathbf{u}_k$:

$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k}\cdot(\mathbf{u}-\mathbf{u}_k), \tag{17}$$

where $\mathbf{x}_k = \mathbf{m}(\mathbf{u}_k)$ is the center of a Gaussian in ray space. The Jacobian $\mathbf{J}_{\mathbf{u}_k}$ is given by the partial derivatives of $\mathbf{m}$ at the point $\mathbf{u}_k$:

$$\mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k). \tag{18}$$

In the following discussion, we are omitting the subscript $\mathbf{u}_k$, hence $\mathbf{m}(\mathbf{u})$ denotes the local affine approximation (17). We substitute $\mathbf{u} = \mathbf{m}^{-1}(\mathbf{x})$ in (14) and apply Equation (10) to map the reconstruction kernels to ray space, yielding the desired expression for $r_k(\mathbf{x})$:

$$\begin{aligned} r_k(\mathbf{x}) &= \frac{1}{|\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}'_k}(\mathbf{m}^{-1}(\mathbf{x})-\mathbf{u}_k) \\ &= \frac{1}{|\mathbf{W}^{-1}||\mathbf{J}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\mathbf{x}-\mathbf{x}_k), \end{aligned} \tag{19}$$

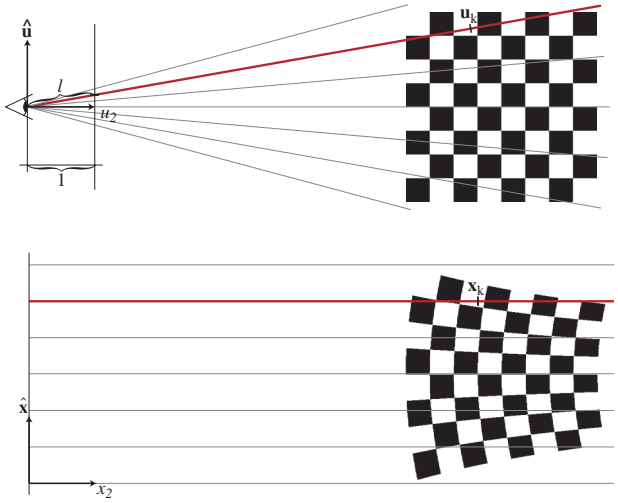


Figure 3: *Transforming the volume from camera to ray space. Top: camera space. Bottom: ray space.*

where $\mathbf{V}_k$ is the variance matrix in ray coordinates. According to (10), $\mathbf{V}_k$ is given by:

$$\begin{aligned} \mathbf{V}_k &= \mathbf{J}\mathbf{V}'_k\,\mathbf{J}^T \\ &= \mathbf{J}\mathbf{W}\mathbf{V}''_k\mathbf{W}^T\mathbf{J}^T. \end{aligned} \tag{20}$$

Note that for uniform or rectilinear data sets, $\mathbf{V}'_k$ has to be computed only once per frame, since $\mathbf{V}''_k$ is the same for all voxels and $\mathbf{W}$ changes only from frame to frame. However, since the Jacobian is different for each voxel position, $\mathbf{V}_k$ has to be recalculated for each voxel. In the case of curvilinear or irregular volumes, each reconstruction kernel has an individual variance matrix $\mathbf{V}''_k$. Our method efficiently handles this situation, requiring only one additional 3×3 matrix multiplication. In contrast, previous techniques [19, 11] cope with elliptical kernels by computing their projected extents in screen space and then establishing a mapping to a circular footprint table. However, this procedure is computationally expensive. It leads to a bad approximation of the integral of the reconstruction kernel as pointed out in [15, 17].

As illustrated in Figure 4, the local affine mapping is exact only for the ray passing through $\mathbf{u}_k$ or $\mathbf{x}_k$, respectively. The figure is exaggerated to show the non-linear effects in the exact mapping. The affine mapping essentially approximates the perspective projection with an oblique orthographic projection. Therefore, parallel lines are preserved, and approximation errors grow with increasing ray divergence. However, the errors do not lead to visual artifacts in general [15], since the fan of rays intersecting a reconstruction kernel has a small opening angle due to the local support of the reconstruction kernels.

A common approach of performing splatting with perspective projection is to map the footprint function onto a *footprint polygon* in camera space in a first step. In the next step, the footprint polygon is projected to screen space and rasterized, resulting in the so-called *footprint image*. As mentioned in [15], however, this requires significant computational effort. In contrast, our framework efficiently performs perspective projection by mapping the volume to ray space, which requires only the computation of the Jacobian and two 3×3 matrix multiplications. For spherical reconstruction kernels, these matrix operations can be further optimized as shown in Section 6.

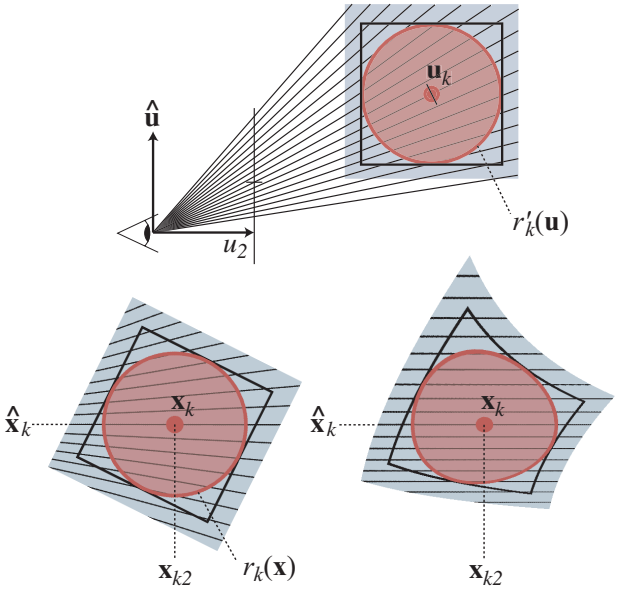


Figure 4: *Mapping a reconstruction kernel from camera to ray space. Top: camera space. Bottom: ray space. Left: local affine mapping. Right: exact mapping.*

4.5 Integration and Band-Limiting

We integrate the Gaussian reconstruction kernel of (19) according to (5), resulting in a Gaussian footprint function q_k:

$$\begin{aligned} q_k(\hat{\mathbf{x}}) &= \int_{\mathbb{R}} \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k, x_2 - x_{k2})\,dx_2 \\ &= \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\hat{\mathbf{V}}_k}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k), \end{aligned} \tag{21}$$

where the 2×2 variance matrix $\hat{\mathbf{V}}_k$ of the footprint function is obtained from $\mathbf{V}_k$ by skipping the last row and column, as shown in (13).

Finally, we choose a Gaussian low-pass filter $h = \mathcal{G}_{\mathbf{V}^h}(\hat{\mathbf{x}})$, where the variance matrix $\mathbf{V}^h \in \mathbb{R}^{2\times 2}$ is typically the identity matrix. With (11), we compute the convolution in (8), yielding the *EWA volume resampling filter*:

$$\begin{aligned} \rho_k(\hat{\mathbf{x}}) &= (q_k \otimes h)(\hat{\mathbf{x}}) \\ &= \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}(\mathcal{G}_{\hat{\mathbf{V}}_k} \otimes \mathcal{G}_{\mathbf{V}^h})(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k) \\ &= \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\hat{\mathbf{V}}_k + \mathbf{V}^h}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k). \end{aligned} \tag{22}$$

5 Reduction from Volume to Surface Reconstruction Kernels

Since our EWA volume resampling filter can handle arbitrary Gaussian reconstruction kernels, we can represent the structure of a volume data set more accurately by choosing the shape of the reconstruction kernels appropriately. For example, we can improve the precision of isosurface rendering by flattening the reconstruction kernels in the direction of the surface normal. We will show below that an infinitesimally flat Gaussian volume kernel is equivalent to a Gaussian surface texture reconstruction kernel [21]. In other words, we can extract and render a surface representation from a volume data set directly by flattening volume reconstruction kernels into surface reconstruction kernels. Our derivation is illustrated in Figure 5.

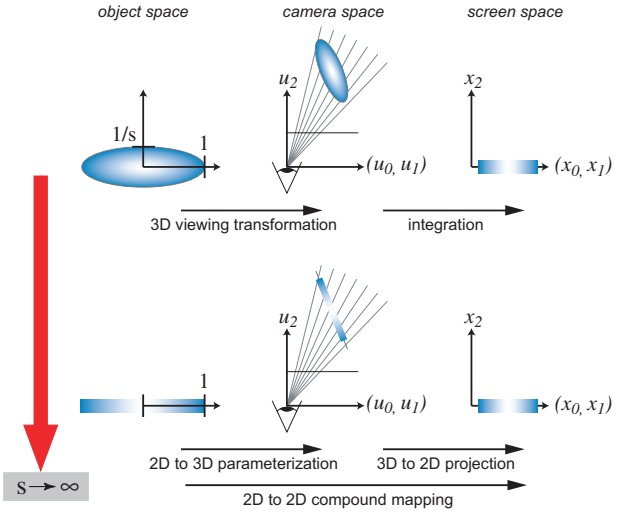


Figure 5: *Reducing a volume reconstruction kernel to a surface reconstruction kernel by flattening the kernel in one dimension. Top: rendering a volume kernel. Bottom: rendering a surface kernel.*

We construct a flattened Gaussian reconstruction kernel in object space by scaling a spherical Gaussian in one direction by a factor $1/s$, hence its variance matrix is:

$$\mathbf{V}'' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{s^2} \end{pmatrix}.$$

A scaling factor $s = 1$ corresponds to a spherical 3D kernel. In the limit, if $s = \infty$, we get a circular 2D kernel.

To render this reconstruction kernel, we first apply a 3D transformation matrix $\mathbf{W}$, which may contain arbitrary modeling transformations concatenated with the viewing transformation. Then we use the local affine approximation of Equation (17) to map the kernel to ray space. The variance matrix $\mathbf{V}$ of the reconstruction kernel in ray space is computed as in (20). We introduce the matrix $\mathbf{T}^{3D}$ to denote the concatenated 3D mapping matrix $\mathbf{T}^{3D} = \mathbf{JW}$ and write $\mathbf{V}$ as:

$$\mathbf{V} = \mathbf{JWV}''\mathbf{W}^T\mathbf{J}^T = \mathbf{T}^{3D}\mathbf{V}''\mathbf{T}^{3D^T}.$$

Hence, the elements v_{ij} of $\mathbf{V}$ are given by:

$$\begin{aligned}
v_{00} &= t_{00}^2 + t_{01}^2 + \frac{t_{02}^2}{s^2} \\
v_{01} = v_{10} &= t_{00}t_{10} + t_{01}t_{11} + \frac{t_{02}t_{12}}{s^2} \\
v_{02} = v_{20} &= t_{00}t_{20} + t_{01}t_{21} + \frac{t_{02}t_{22}}{s^2} \\
v_{11} &= t_{10}^2 + t_{11}^2 + \frac{t_{12}^2}{s^2} \\
v_{12} = v_{21} &= t_{10}t_{20} + t_{11}t_{21} + \frac{t_{12}t_{22}}{s^2} \\
v_{22} &= t_{20}^2 + t_{21}^2 + \frac{t_{22}^2}{s^2},
\end{aligned}$$

where we denote an element of $\mathbf{T}^{3D}$ by t_{ij}.

We compute the 2D Gaussian footprint function by integrating the reconstruction kernel. According to (13), its 2D variance matrix is obtained by skipping the third row and column in $\mathbf{V}$. As s approaches infinity, we therefore get the following 2D variance matrix $\hat{\mathbf{V}}$:

$$\hat{\mathbf{V}} = \begin{pmatrix} t_{00}^2 + t_{01}^2 & t_{00}t_{10} + t_{01}t_{11} \\ t_{00}t_{10} + t_{01}t_{11} & t_{10}^2 + t_{11}^2 \end{pmatrix}. \qquad (23)$$

Conveniently, the 2D variance matrix can be factored into a 2D mapping $\mathbf{T}^{2D}$, which is obtained from the 3D mapping matrix by skipping the third row and column:

$$\hat{\mathbf{V}} = \mathbf{T}^{2D}\mathbf{T}^{2D^T} = \begin{pmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{pmatrix}\begin{pmatrix} t_{00} & t_{10} \\ t_{01} & t_{11} \end{pmatrix}. \qquad (24)$$

Let us now analyze the 2D mapping matrix $\mathbf{T}^{2D}$. First, we need an explicit expression for the Jacobian $\mathbf{J}$ of the projective mapping. Using (15) and (18), it is given by:

$$\mathbf{J} = \begin{pmatrix} 1/u_2 & 0 & -u_0/u_2^2 \\ 0 & 1/u_2 & -u_1/u_2^2 \\ u_0/l' & u_1/l' & u_2/l' \end{pmatrix}, \qquad (25)$$

where $l' = \|(u_0, u_1, u_2)^T\|$. With $\mathbf{T}^{3D} = \mathbf{JW}$, we use the first two rows of $\mathbf{J}$ and the first two columns of $\mathbf{W}$ to factor $\mathbf{T}^{2D}$ into:

$$\mathbf{T}^{2D} = \begin{pmatrix} 1/u_2 & 0 & -u_0/u_2^2 \\ 0 & 1/u_2 & -u_1/u_2^2 \end{pmatrix}\begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{pmatrix},$$

where w_{ij} denotes an element of $\mathbf{W}$. This can be interpreted as a concatenation of a 2D to 3D with a 3D to 2D mapping, resulting in a compound 2D to 2D mapping similar as in conventional texture mapping [3]. We illustrate this process schematically in Figure 5 and more intuitively in Figure 6. The first stage is a parameterization of a 3D plane. It maps a circular 2D texture kernel onto a plane defined by the two vectors $(w_{00}, w_{10}, w_{20})^T$ and $(w_{01}, w_{11}, w_{21})^T$ in 3D camera space, resulting in an ellipse. The second stage is an oblique parallel projection with an additional scaling factor $1/u_2$, which is the local affine approximation of the perspective projection. Finally, combining the projected ellipse with a low-pass filter as in Equation (8) yields a texture filter that is equivalent to Heckbert's EWA filter [4]. This is the same result as we derive in [21]. We compare splatting with volumetric kernels and splatting with surface kernels in Section 7.

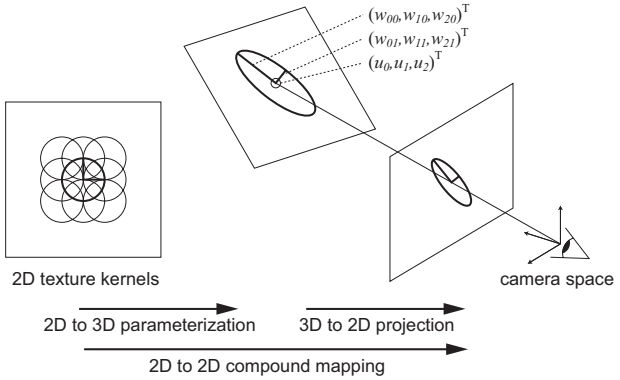


Figure 6: *Rendering surface kernels.*

6 Implementation

We implemented a volume rendering algorithm based on the EWA splatting equation. Our implementation is embedded in the VTK (visualization toolkit) framework [16]. We did not optimize our code for rendering speed. We use a sheet buffer to first accumulate splats from planes in the volume that are most parallel to the projection plane [19]. In a second step, the final image is computed

by compositing the sheets back to front. Shading is performed using the gradient estimation functionality provided by VTK and the Phong illumination model.

We summarize the main steps which are required to compute the EWA splat for each voxel:

```
1: for each voxel k {
2:  compute camera coords. u[k];
3:  compute the Jacobian J;
4:  compute the variance matrix V[k];
5:  project u[k] to screen coords. x_hat[k];
6:  setup the resampling filter rho[k];
7:  rasterize rho[k];
8: }
```

First, we compute the camera coordinates $\mathbf{u}_k$ of the current voxel k by applying the viewing transformation to the voxel center. Using $\mathbf{u}_k$, we then evaluate the Jacobian $\mathbf{J}$ as given in Equation (25). In line 4, we transform the Gaussian reconstruction kernel from object to ray space. This transformation is implemented by Equation (20), and it results in the 3×3 variance matrix $\mathbf{V}_k$ of the Gaussian in ray space. Remember that $\mathbf{W}$ is the rotational part of the viewing transformation, hence it is typically orthonormal. Moreover, for spherical kernels, $\mathbf{V}_k''$ is the identity matrix. In this case, evaluation of Equation (20) can be simplified significantly. Next, we project the voxel center from camera space to the screen by performing a perspective division on $\mathbf{u}_k$. This yields the 2D screen coordinates $\hat{\mathbf{x}}_k$. Now we are ready to setup the resampling filter ρ_k according to Equation (22). Its variance matrix is derived from $\mathbf{V}_k$ by omitting the third row and column, and adding a 2×2 identity matrix for the low-pass filter. Moreover, we compute the determinants $1/|\mathbf{J}^{-1}|$ and $1/|\mathbf{W}|^{-1}$ that are used as normalization factors.

Finally, we rasterize the resampling filter in line 7. As can be seen from the definition of the elliptical Gaussian (9), we also need the inverse of the variance matrix, which is called the *conic matrix*. Let us denote the 2×2 conic matrix of the resampling filter by $\mathbf{Q}$. Furthermore, we define the radial index function

$$r(\bar{\mathbf{x}}) = \bar{\mathbf{x}}^T \mathbf{Q} \bar{\mathbf{x}} \quad \text{where} \quad \bar{\mathbf{x}} = (\bar{x}_0, \bar{x}_1)^T = \hat{\mathbf{x}} - \hat{\mathbf{x}}_k.$$

Note that the contours of the radial index, i.e., $r = const.$ are concentric ellipses. For circular kernels, r is the squared distance to the circle center. The exponential function in (9) can now be written as $e^{-\frac{1}{2}r}$. We store this function in a 1D lookup table. To evaluate the radial index efficiently, we use finite differencing. Since r is biquadratic in $\bar{\mathbf{x}}$, we need only two additions to update r for each pixel. We rasterize r in a rectangular, axis aligned bounding box centered around $\hat{\mathbf{x}}_k$ as illustrated in Figure 7. Typically, we use a threshold $c = 4$ and evaluate the Gaussian only if $r(\bar{\mathbf{x}}) < c$. Heckbert provides pseudo-code of the rasterization algorithm in [4].

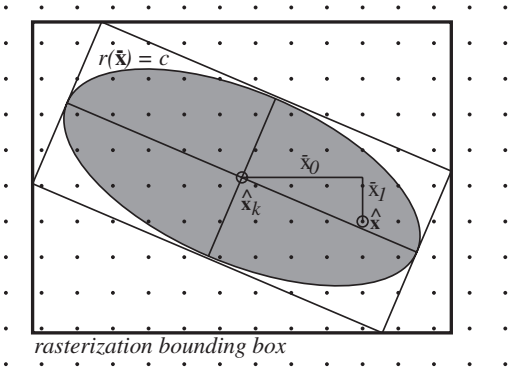


Figure 7: *Rasterizing the resampling filter.*

7 Results

The EWA resampling filter has a number of useful properties, as illustrated in Figure 8. When the mapping from camera to ray space minifies the volume, size and shape of the resampling filter are dominated by the low-pass filter, as in the left column of Figure 8. In the middle column, the volume is magnified and the resampling filter is dominated by the reconstruction kernel. Since the resampling filter unifies a reconstruction kernel and a low-pass filter, it provides a smooth transition between magnification and minification. Moreover, the reconstruction kernel is scaled anisotropically in situations where the volume is stretched in one direction and shrinked in the other, as shown in the right column. In the bottom row, we show the filter shapes resulting from uniformly scaling the reconstruction kernel to avoid aliasing, as proposed by Swan et al. [17]. Essentially, the reconstruction kernel is enlarged such that its minor radius is at least as long as the minor radius of the low-pass filter. For spherical reconstruction kernels, or circular footprint functions, this is basically equivalent to the EWA resampling filter. However, for elliptical footprint functions, uniform scaling leads to overly blurred images in the direction of the major axis of the ellipse.

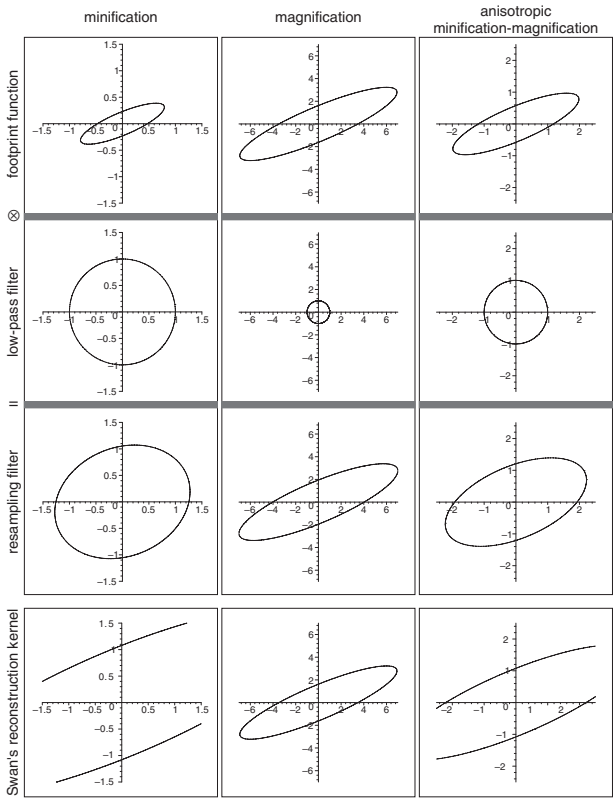


Figure 8: *Properties of the EWA resampling filter*

We compare our method to Swan's method in Figure 8 (see colorplate). The images on the left were rendered with EWA volume splats, those on the right with Swan's uniformly scaled kernels. We used a square zebra texture with *x* and *y* dimensions of 1024×512 in the first row, and 1024×256 in the second row. This leads to elliptical reconstruction kernels with a ratio between the length of the major and minor radii of 2 to 1 and 4 to 1, respectively. Clearly, the EWA filter produces a crisper image and at the same time does not exhibit aliasing artifacts. As the ratio between the major and minor radii of the reconstruction kernels increases, the difference to Swan's method becomes more pronounced. For strongly anisotropic kernels, Swan's uniform scaling produces excessively

blurred images, as shown on the right in Figure 8 . Each frame took approximately 6 seconds to render on an 866 MHz PIII processor.

In Figure 9 (see colorplate), we compare EWA splatting using volume kernels on the left to surface reconstruction kernels on the right. The texture size is 512×512 in *x* and *y* direction. Typically, the perspective projection of a spherical kernel is almost a circle. Therefore, rendering with volume kernels does not exhibit anisotropic texture filtering and produces textures that are slightly too blurry, similar to isotropic texture filters such as trilinear mipmapping. On the other hand, splatting surface kernels is equivalent to EWA texture filtering. Circular surface kernels are mapped to ellipses, which results in high image quality because of anisotropic filtering.

In Figure 10 (see colorplate), we show a series of volume renderings of the UNC CT scan of a human head ($256 \times 256 \times 225$), the UNC engine ($256 \times 256 \times 110$), and the foot of the visible woman dataset ($152 \times 261 \times 220$). The texture in the last example is rendered using EWA surface splatting, too. The images illustrate that our algorithm correctly renders semitransparent objects as well. The skull of the UNC head, the bone of the foot, and the iso-surface of the engine were rendered with flattened surface splats oriented perpendicular to the volume gradient. All other voxels were rendered with EWA volume splats. Each frame took approximately 11 seconds to render on an 866 MHz PIII processor.

8 Conclusion and Future Work

We present a new splat primitive for volume rendering, called the EWA volume resampling filter. Our primitive provides high quality antialiasing for splatting algorithms, combining an elliptical Gaussian reconstruction kernel with a Gaussian low-pass filter. We use a novel approach of computing the footprint function. Exploiting the mathematical features of 2D and 3D Gaussians, our framework efficiently handles arbitrary elliptical reconstruction kernels and perspective projection. Therefore, our primitive is suitable to render regular, rectilinear, curvilinear, and irregular volume data sets. Finally, we derive a formulation of the EWA surface reconstruction kernel, which is equivalent to Heckbert's EWA texture filter. Hence we call our primitive *universal*, facilitating the reconstruction of surface and volume data.

We have not yet investigated whether other kernels besides elliptical Gaussians may be used with this framework. In principle, a resampling filter could be derived from any function that allows the analytic evaluation of the operations described in Section 4.2 and that is a good approximation of an ideal low-pass filter.

To achieve interactive frame rates, we are currently investigating the use of graphics hardware to rasterize EWA splats as texture mapped polygons. We also plan to use sheet-buffers that are parallel to the image plane to eliminate popping artifacts. To render non-rectilinear datasets we are investigating fast back-to-front sorting algorithms. Furthermore, we want to experiment with our splat primitive in a post-shaded volume rendering pipeline. The derivative of the EWA resampling filter could be used as a band-limited gradient kernel, hence avoiding aliasing caused by shading for noisy volume data. Finally, we want to exploit the ability of our framework to render surface splats. In conjunction with voxel culling algorithms we believe it is useful for real-time iso-surface rendering.

9 Acknowledgments

Many thanks to Lisa Sobierajski Avila for her help with our implementation of EWA volume splatting in vtk. We would also like to thank Paul Heckbert for his encouragement and helpful comments. Thanks to Chris Wren for his supporting role in feeding us, and to Jennifer Roderick and Martin Roth for proofreading the paper.

References

[1] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *1994 Workshop on Volume Visualization*, pages 91–98. Washington, DC, October 1994.

[2] A. Van Gelder and K. Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. In *ACM/IEEE Symposium on Volume Visualization*, pages 23–30. San Francisco, CA, October 1996.

[3] P. Heckbert. Survey of Texture Mapping. *IEEE Computer Graphics & Applications*, 6(11):56–67, November 1986.

[4] P. Heckbert. *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, University of California at Berkeley, Department of Electrical Engineering and Computer Science, June 17 1989.

[5] James T. Kajiya and Brian P. Von Herzen. Ray Tracing Volume Densities. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):165–174, July 1984. Held in Minneapolis, Minnesota.

[6] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp factorization of the Viewing Transform. In *Computer Graphics*, Proceedings of SIGGRAPH 94, pages 451–457. July 1994.

[7] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. In *Computer Graphics*, SIGGRAPH '91 Proceedings, pages 285–288. Las Vegas, NV, July – August 1991.

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

[9] L. Lippert and M. H. Gross. Fast Wavelet Based Volume Rendering by Accumulation of Transparent Texture Maps. *Computer Graphics Forum*, 14(3):431–444, August 1995. ISSN 1067-7055.

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

[11] X. Mao, L. Hong, and A. Kaufman. Splatting of Curvilinear Volumes. In *IEEE Visualization '95 Proc.*, pages 61–68. October 1995.

[12] N. Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[13] K. Mueller, T. Moeller, and R. Crawfis. Splatting Without the Blur. In *Proceedings of the 1999 IEEE Visualization Conference*, pages 363–370. San Francisco, CA, October 1999.

[14] Klaus Mueller and Roger Crawfis. Eliminating Popping Artifacts in Sheet Buffer-Based Splatting. *IEEE Visualization '98*, pages 239–246, October 1998. ISBN 0-8186-9176-X.

[15] Klaus Mueller and Roni Yagel. Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach. *IEEE Visualization '96*, pages 65–72, October 1996. ISBN 0-89791-864-9.

[16] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall, second edition, 1998.

[17] J. E. Swan, K. Mueller, T. Möller, N. Shareef, R. Crawfis, and R. Yagel. An Anti-Aliasing Technique for Splatting. In *Proceedings of the 1997 IEEE Visualization Conference*, pages 197–204. Phoenix, AZ, October 1997.

[18] L. Westover. Interactive Volume Rendering. In C. Upson, editor, *Proceedings of the Chapel Hill Workshop on Volume Visualization*, pages 9–16. University of North Carolina at Chapel Hill, Chapel Hill, NC, May 1989.

[19] L. Westover. Footprint Evaluation for Volume Rendering. In *Computer Graphics*, Proceedings of SIGGRAPH 90, pages 367–376. August 1990.

[20] C. Wittenbrink, T. Malzbender, and M. Goss. Opacity-Weighted Color Interpolation For Volume Sampling. *IEEE Symposium on Volume Visualization, 1998*, pages 431–444. ISBN 0-8186-9180-8.

[21] M. Zwicker, H. Pfister., J. Van Baar, and M. Gross. Surface Splatting. In *Computer Graphics*, SIGGRAPH 2001 Proceedings. Los Angeles, CA, July 2001.

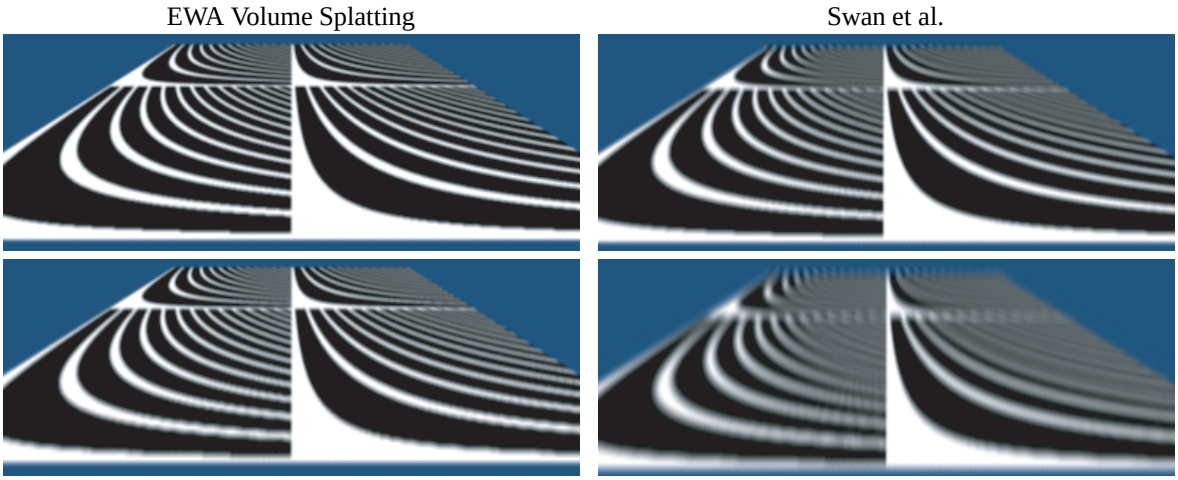


Figure 8: *Comparison between EWA volume splatting and Swan et al. Top row:* $1024 \times 512 \times 3$ *volume texture. Bottom row:* $1024 \times 256 \times 3$ *volume texture. The image resolution is* 400×150 *pixels.*

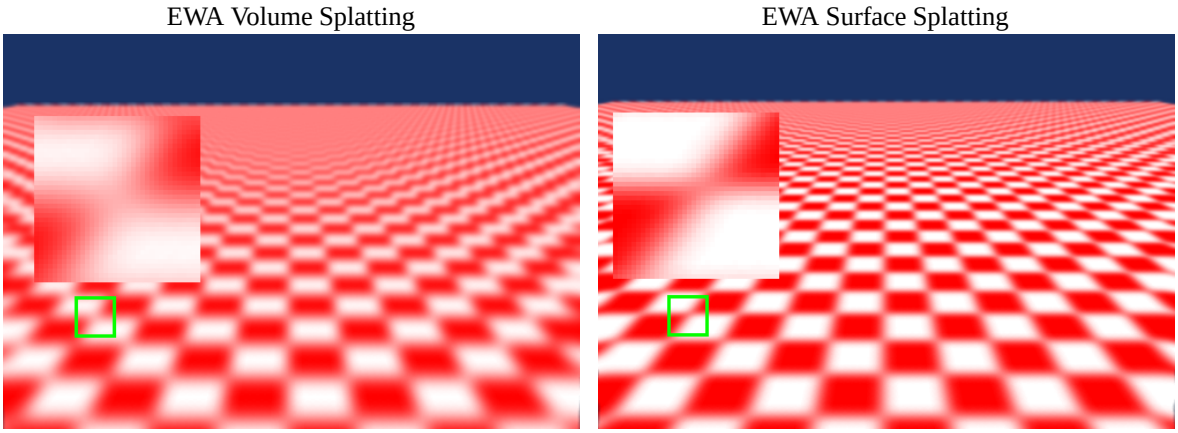
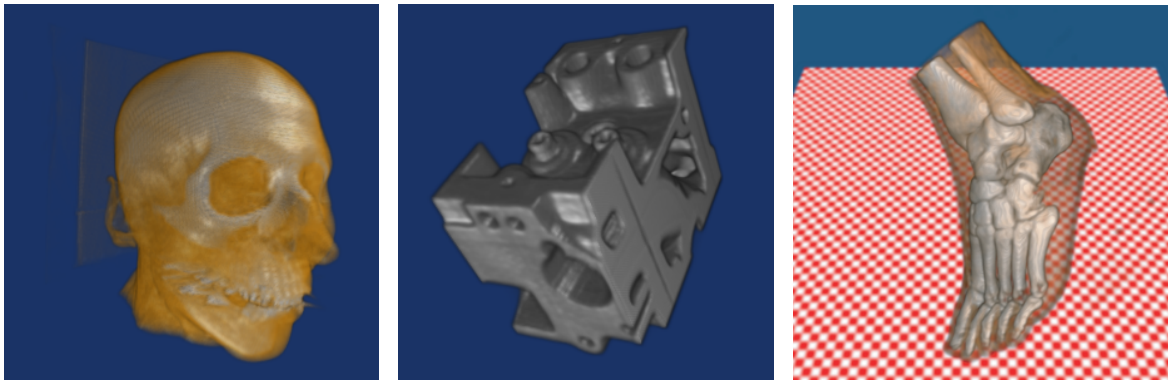


Figure 9: *EWA volume splatting versus EWA surface splatting;* $512 \times 512 \times 3$ *volume texture. The image resolution is* 500×342 *pixels.*



(a) UNC Head (b) UNC Engine (c) Visible Woman Foot

Figure 10: *Semitransparent objects rendered using EWA volume splatting. The skull of the UNC head, the iso-surface of the engine, and the bone of the foot are rendered with flattened surface splats.*