Perspective Accurate Splatting

Matthias Zwicker
Massachusetts Institute of Technology

Jussi Räsänen
Hybrid Graphics, Helsinki University of Technology

Mario Botsch
TU Aachen

Carsten Dachsbacher
Universität Erlangen-Nürnberg

Mark Pauly
Stanford University

Abstract

We present a novel algorithm for accurate, high quality point rendering, which is based on the formulation of splatting using homogeneous coordinates. In contrast to previous methods, this leads to perspective correct splat shapes, avoiding artifacts such as holes caused by the affine approximation of the perspective projection. Further, our algorithm implements the EWA resampling filter, hence providing high image quality with anisotropic texture filtering. We also present an extension of our rendering primitive that facilitates the display of sharp edges and corners. Finally, we describe an efficient implementation of the entire point rendering pipeline using vertex and fragment programs of current GPUs.

Key words: point rendering, conic sections and projective mappings, texture filtering, graphics hardware

1 Introduction

Because of the growing availability and wide use of 3D scanning technology [10, 14], point-sampled surface data has become more important and attracted increasing interest in computer graphics. 3D scanning devices often produce huge volumes of point data that are difficult to process, edit, and visualize. Instead of reconstructing consistent triangle meshes or higher order surface representations from the acquired data, point-based approaches work directly with the sampled points without requiring the computation of connectivity information or imposing constraints on the sample distribution. Point-based methods have proven to be efficient for processing, editing, and rendering such data [17, 20, 23].

Point rendering is particularly interesting for highly complex models, whose triangle representation requires millions of tiny primitives. The projected area of those triangles is often less than a few pixels, resulting in inefficient rendering because of the overhead for triangle setup and making point primitives a promising alternative. Until recently, limited programmability has hampered the implementation of point-based rendering algorithms on graphics hardware. However, with the current generation of graphics processors (GPUs), it is now possible to control a large part of the rasterization process. In this paper we present a point rendering method that is completely implemented on the GPU by exploiting such capabilities. In addition, our technique is derived from a novel formulation of the splatting process using homogeneous coordinates, which facilitates accurate, high quality point-rendering.

Our approach is based on Gaussian resampling filters as introduced by Heckbert [7]. Resampling filters combine a reconstruction filter and a low-pass filter into a single filter kernel, which leads to rendering algorithms with high quality antialiasing capabilities. Using an *affine approximation* of a general projective mapping, Heckbert derived a Gaussian resampling filter, known as the *EWA (elliptical weighted average) filter*, which can be computed efficiently. While Heckbert originally developed resampling filters in the context of texture mapping [4], the technique has been reformulated and applied to point rendering recently [24]. Here, a resampling filter in *image space*, also called a *splat*, is computed and rasterized for each point. Using resampling filters for point rendering, most sampling artifacts such as holes or aliasing can be effectively avoided.

The contributions of this paper improve upon previous point rendering techniques in several ways. First, we present a novel approach to express the splat shape in image space using homogeneous coordinates. Unlike previous methods, the splat shape we compute is the exact perspective projection of an elliptical reconstruction kernel in object space. Hence, the proposed method leads to more accurate reconstruction of surfaces in image space and avoids holes even under extreme perspective projections. Note that although our method computes the perspective correct splat shape, the resampling kernel is still based on an affine approximation of the perspective projection. Further, we describe a direct implementation of our algorithm using current programmable graphics hardware. Instead of rendering each splat as a quad or a triangle as proposed, e.g., by Ren et al. [22], we use OpenGL point primitives and fragment programs to rasterize the resampling filter. Hence, we avoid the overhead of sending several vertices to the graphics processor

for each point. Our implementation evaluates the EWA resampling filter for arbitrary elliptical object space reconstruction filters, while previous approaches only allowed circular kernels and computed approximations of the resampling filter [1]. Finally, we describe an extension to elliptical splats by adding a clip line to the splat. We demonstrate how this method enables the rendering of objects with sharp features at little additional cost.

2 Related Work

The use of points as a display primitive was first proposed in the seminal work by Levoy and Whitted [11]. In their report, they identified the fundamental issues of point rendering, such as filtering and surface reconstruction. Grossman and Dally [5] designed a point-based rendering pipeline that used an efficient image space surface reconstruction technique. Their approach was improved by Pfister et al. [21] by adding a hierarchical object representation and texture filtering. Focusing on the visualization of large data sets acquired using 3D scanning, Rusinkiewicz et al. developed the QSplat system [23], which leveraged graphics hardware for point rendering.

A principled analysis of the sampling issues arising in point rendering was first provided by Zwicker et al. [24]. This work is based on the concept of resampling filters introduced by Heckbert [7]. Heckbert showed how to unify a Gaussian reconstruction and band-limiting step in a single Gaussian resampling kernel. Point rendering with Gaussian resampling kernels, also called *EWA splatting*, has provided the highest image quality so far, with antialiasing capabilities similar to anisotropic texture filtering [15]. To reduce the computational complexity of EWA splatting, an approximation using look-up tables has been presented by Botsch et al. [2].

Several authors have leveraged the computational power and programmability of current GPUs [13, 12] for further increasing the performance of EWA splatting. Ren et al. proposed to represent the resampling filter for each point by a quad [22]. However, this has the disadvantage that the data sent to the GPU is multiplied by a factor of four. A more efficient approach based on point sprites was introduced by Botsch et al. [1]. This method is restricted to circular reconstruction kernels and uses an approximation of EWA splatting. Our technique is similar in that we also use point primitives for rasterizing splats. However, we implement exact EWA splatting and handle arbitrary elliptical reconstruction kernels. In addition, our approach is based on a novel formulation of splatting using homogeneous coordinates, which resembles the technique described by Olano and Greer [16] for rasterizing triangles. A GPU implementation of EWA splatting was also presented by Guennebaud et al. [6].

3 Point Rendering as a Resampling Process

To provide the necessary background for our contributions described in Section 4, we start by summarizing the underlying techniques introduced by Heckbert [7] and Zwicker et al. [24]. We first define the notion of point-sampled surfaces as nonuniformly sampled signals and explain how these are rendered by reconstructing a continuous signal in image space. Then, we briefly review the concept of resampling filters and show how it is applied in the context of point rendering.

A point-based surface consists of a set of nonuniformly distributed samples of a surface, hence we interpret it as a *nonuniformly sampled signal*. To continuously reconstruct this signal, we associate a 2D reconstruction kernel $r_k(\mathbf{u})$ with each sample point $\mathbf{p}_k$. These kernels are defined in a local tangent frame with coordinates $\mathbf{u} = (u, v)$ at the point $\mathbf{p}_k$, as illustrated on the left in Figure 1. The tangent frames and the parameters of the reconstruction kernels can be computed from the point set as described by Pauly et al. [18].

The surface is rendered by reconstructing it in image space. For now we focus on the reconstruction of the surface color and denote a color sample at $\mathbf{p}_k$ by f_k. Rendering is achieved by projectively mapping the reconstruction kernels from their local tangent frames to the image plane and building the weighted sum

$$g(\mathbf{x}) = \sum_k f_k r_k(\mathbf{M}_k^{-1}(\mathbf{x})) = \sum_k f_k r'_k(\mathbf{x}), \qquad (1)$$

where g is the rendered surface, $\mathbf{x}$ are 2D image space coordinates, and $\mathbf{M}_k$ is the 2D-to-2D projective mapping from the local tangent frame of point $\mathbf{p}_k$ to image space. In addition, reconstruction kernels mapped to image space are denoted by $r'_k(\mathbf{x})$. This is illustrated in Figure 1.

Since the reconstructed signal $g(\mathbf{x})$ contains arbitrarily high frequencies, sampling it at the output pixel grid leads to aliasing artifacts. To avoid such artifacts, Heckbert [7] introduced the concept of *resampling filters*. In this approach the reconstruction step is combined with a prefiltering step, hence band-limiting the signal to the Nyquist frequency of the pixel grid. We include prefiltering in our rendering procedure by convolving Equation 1 with a low-pass filter h:

$$\begin{aligned} g'(\mathbf{x}) &= \sum_k f_k r'_k(\mathbf{x}) \otimes h(\mathbf{x}) \\ &= \sum_k f_k \rho_k(\mathbf{x}), \end{aligned} \qquad (2)$$

where the *resampling kernels* ρ_k unify the reconstruction kernels in image space and the prefilter. Note that

because the resampling filters do not form a partition of unity, Equation 2 is usually normalized by dividing through the sum of resampling filters (see also Section 6).

To derive a practical resampling filter, Heckbert chose Gaussians as reconstruction and low-pass filters. A 2D Gaussian is defined as

$$g_{\mathbf{V}}(\mathbf{x}) = \frac{|\mathbf{V}^{-1}|^{1/2}}{2\pi} e^{-\frac{1}{2}\mathbf{x}\mathbf{V}^{-1}\mathbf{x}^T},$$

where $\mathbf{V}$ is a 2×2 *variance matrix* and $\mathbf{x}$ is a 1×2 row vector. We denote Gaussian reconstruction and low-pass filters by $r_k = g_{\mathbf{R}_k}$ and $h = g_{\mathbf{H}}$. In particular, Heckbert showed that the resampling filter is again a Gaussian if the projective mappings $\mathbf{M}_k$ are approximated by affine mappings. In this case, the reconstruction kernel in screen space r'_k is a Gaussian

$$r'_k(\mathbf{x}) = \frac{1}{|\mathbf{R}'_k|^{1/2}} g_{\mathbf{R}'_k}(\mathbf{x}),$$

with a new variance matrix $\mathbf{R}'_k$. One of our contributions, presented in Section 4, is a novel approach to compute $\mathbf{R}'_k$. The variance matrix $\mathbf{H}$ of the low-pass filterd is typically an identity matrix. The resampling filter is then given by

$$\rho_k(\mathbf{x}) = \frac{1}{|\mathbf{R}'_k|^{1/2}} g_{\mathbf{R}'_k + \mathbf{H}}(\mathbf{x}). \tag{3}$$

This has also been called the *EWA resampling filter*; we refer the reader to [7, 24] for more details on its derivation.

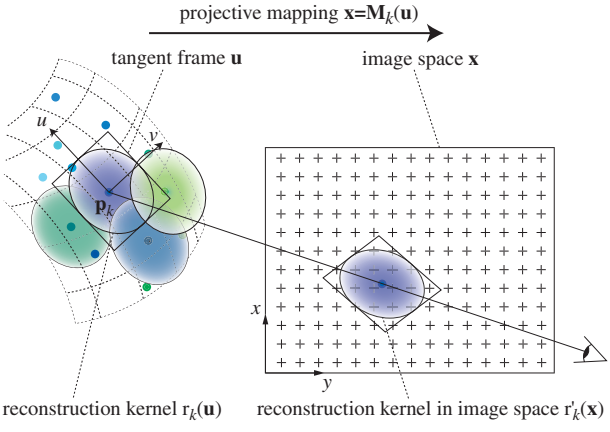

Figure 1: Point-based surfaces are rendered by mapping the reconstruction kernels from the local tangent frames to image space.

4 Perspective Accurate Splatting Using Homogeneous Coordinates

Similar to previous techniques, our approach is based on Gaussian resampling filters as in Equation 3. However, we introduce a new approach to compute the perspective correct shape of the kernels, where in previous techniques this shape is an affine approximation. Our method is based on the formulation of the 2D projective mappings from local tangent frames to image space using homogeneous coordinates, as described in Section 4.1. We then review the definition of conic sections using homogeneous coordinates in Section 4.2 and show how to compute projective mappings of conics in Section 4.3. Finally, we use these results in Section 4.4 to derive Gaussian resampling kernels with perspective accurate shapes.

4.1 Homogeneous Coordinates and Projective Mappings

As explained in the previous section, reconstruction kernels are defined on local tangent planes and mapped to image space by projective mappings during rendering. With homogeneous coordinates, a general 2D-to-2D projective mapping from a source to a destination plane may be written as $\mathbf{x} = \mathbf{u}\mathbf{M}^\star$. Here $\mathbf{x} = (xz, yz, z)$ and $\mathbf{u} = (uw, vw, w)$ with $z, w \neq 0$ are homogeneous points in the source and destination planes, and $\mathbf{M}^\star$ is a 3×3 mapping matrix. Note that the inverse of a projective mapping can be formed by inverting the mapping matrix, and the inverse is again a projective mapping.

To determine the mapping matrix for our application, let us define a *tangent plane* in 3D with coordinates (x, y, z) by a point $\mathbf{p}_k$ and two tangent vectors $\mathbf{t_u}$ and $\mathbf{t_v}$. The tangent vectors form the basis of a 2D coordinate system on the plane, whose coordinates we denote by u and v. We can express points $\mathbf{p} = (p_x, p_y, p_z)$ on the plane in matrix form:

$$\mathbf{p} = (u, v, 1) \begin{bmatrix} \mathbf{t_u} \\ \mathbf{t_v} \\ \mathbf{p_k} \end{bmatrix} = (u, v, 1)\, \mathbf{M}_k. \tag{4}$$

Now we specify the *image plane* in 3D by a point $(0, 0, 1)$ that lies on the plane and tangent vectors $(1, 0, 0)$ and $(0, 1, 0)$. Note that we can always transform both planes such that the image plane has this position. In fact, this corresponds to the transformation of the scene geometry from world to camera space. The projection of the point $\mathbf{p}$ from the tangent plane through the origin $(0, 0, 0)$ onto the image plane is now

$$(x, y, 1) = \left(\frac{p_x}{p_z}, \frac{p_y}{p_z}, 1\right).$$

This is equivalent to regarding $\mathbf{M}_k$ in Equation 4 as the mapping matrix $\mathbf{M}^\star$ of a projective mapping and $\mathbf{p}$ as a homogeneous point. Hence, the projective mapping from the tangent plane to image space is given by $\mathbf{x} = \mathbf{u}\mathbf{M}_k$, with $\mathbf{x}$ and $\mathbf{u}$ being homogeneous points in the image

plane and the tangent plane, respectively. Further, the inverse of the mapping is $\mathbf{u} = \mathbf{x}\,\mathbf{M}_k^{-1}$.

4.2 Implicit Conics in Homogeneous Coordinates

Conics are central to our technique since the isocontours of the Gaussian kernels that we use are ellipses, which are special cases of general conics. The implicit form of a general conic is

$$\phi(x,y) = Ax^2+2Bxy+Cy^2+2Dx+2Ey-F = 0 \quad (5)$$

Because of its implicit form, all scalar multiples of Equation 5 are equivalent and we can assume that $A >= 0$. The equation describes an ellipse if the discriminant $\Delta = AC - B^2$ is greater than 0, a parabola if $\Delta = 0$ and a hyperbola if $\Delta < 0$. Using homogeneous coordinates we can express a general conic in matrix form:

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} A & B & D \\ B & C & E \\ D & E & -F \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{x}\mathbf{Q}_h\mathbf{x}^T = 0. \quad (6)$$

When $D = E = 0$, the *center* of the conic is at the origin. The resulting form

$$Ax^2 + 2Bxy + Cy^2 = F$$

is called the *central conic* [1]. A central conic can be expressed in the matrix form

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{x}\mathbf{Q}\mathbf{x}^T = F,$$

where $\mathbf{Q}$ is the *conic matrix*.

A general conic can be transformed into a central conic by writing the general conic with the center offset to $\mathbf{x}_t = (x_t, y_t)$:

$$A(x+x_t)^2 + 2B(x+x_t)(y+y_t) + C(y+y_t)^2 + 2D(x+x_t) + 2E(y+y_t) - F = 0.$$

To determine the offset $\mathbf{x}_t$, we require that the terms involving the first degree of x and y are 0. By solving the resulting system of two equations we find

$$\mathbf{x}_t = (x_t, y_t) = \left(\frac{(BE - CD)}{\Delta}, \frac{(BD - AE)}{\Delta} \right), \quad (7)$$

and the resulting central conic is

$$Ax^2 + 2Bxy + Cy^2 = F - Dx_t - Ey_t.$$

[1] In some texts the central conic is called the *canonical conic*.

Note that for parabolas, for which $\Delta = 0$, the center is at infinity.

As will be described in Section 6, we rasterize implicit conics by testing at each pixel in the image plane whether it is inside or outside the conic. To minimize the number of pixels that are tested, we compute a tight *axis aligned bounding box* of the conic. The extremal x values x_{max} and x_{min} of the conic (Equation 5) are given by the constraint that its partial derivative in the y direction $\frac{\partial \phi}{\partial y}$ vanishes, while the extremal y values are taken at the point where the partial derivative in the x direction $\frac{\partial \phi}{\partial x}$ vanishes. Hence by substituting these constraints

$$\frac{\partial \phi}{\partial y} = 2Bx + 2Cy + 2E = 0$$
$$\frac{\partial \phi}{\partial x} = 2Ax + 2By + 2D = 0$$

into Equation 5, we find the bounds

$$x_{max}, x_{min} = x_t \pm \sqrt{\frac{C(F - Dx_t - Ey_t)}{\Delta}} \quad (8)$$

$$y_{max}, y_{min} = y_t \pm \sqrt{\frac{A(F - Dx_t - Ey_t)}{\Delta}}. \quad (9)$$

4.3 Projective Mappings of Conics

As we saw in Section 4.1, we can express a 2D-to-2D projective mapping in matrix form as $\mathbf{x} = \mathbf{u}\mathbf{M}$, where $\mathbf{x}$ and $\mathbf{u}$ are homogeneous coordinates. To apply this mapping to a conic $\mathbf{u}\mathbf{Q}_h\mathbf{u}^T = 0$, we substitute $\mathbf{u} = \mathbf{x}\mathbf{M}^{-1}$. This yields another conic $\mathbf{x}\mathbf{Q}'_h\mathbf{x}^T = 0$, where

$$\mathbf{Q}'_h = \mathbf{M}^{-1}\mathbf{Q}_h\mathbf{M}^{-1^T} = \begin{bmatrix} a & b & d \\ b & c & e \\ d & e & -f \end{bmatrix}.$$

Hence we have derived the widely known fact that conics are *closed* under projective mappings.

Using Equation 7, we now transform the projectively mapped conic into a central conic, yielding

$$(\mathbf{x} - \mathbf{x}_t)\mathbf{Q}''(\mathbf{x} - \mathbf{x}_t)^T \leq f - dx_t - ey_t \quad (10)$$

where

$$\mathbf{Q}'' = \begin{bmatrix} a & b \\ b & c \end{bmatrix}.$$

Note that although we applied a *projective* mapping to the conic from Equation 6, the conic in Equation 10 is expressed as an *affine* mapping of the original conic. However, only the points on the *conic curve* are mapped to the same positions by the two mappings. Otherwise, the mappings do not correspond. This is due to the fact that the transformation to the central conic form is based on the properties of conics, neglecting the properties of projective mappings.

4.4 Application to Gaussian Filters

We now apply the technique described above to derive Gaussian resampling filters with perspective accurate splat shapes. To this end, we need to approximate the projective mappings of the reconstruction kernels from the tangent planes to image space by *affine mappings* (see Section 3). In previous work [24] the approximation was chosen such that it is exact at the center of the reconstruction kernels. In contrast, we choose the affine mapping such that it matches the projective mapping of a *conic isocontour* of the Gaussian kernels.

In practice, Gaussians are truncated to a finite support. I.e., the reconstruction kernels $g_{\mathbf{R}_k}(\mathbf{u})$ are evaluated only within conic isocontours $\mathbf{u}\mathbf{R}_k^{-1}\mathbf{u}^T < F_g^2$, where F_g is a user specified *cutoff value* (typically in the range $1 < F_g < 2$). With homogeneous coordinates, an isocontour can also be expressed as

$$\begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_k & 0 \\ 0 & -F_g^2 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{u}\mathbf{Q}_h\mathbf{u}^T = 0.$$

Remember that Equation 10 expresses a projective mapping of this conic using an affine mapping; hence we use it as our affine approximation for the projective mapping $\mathbf{x} = \mathbf{u}\mathbf{M}_k$ from local tangent frames to image space. Note that we need to scale Equation 10 to get the isocontour with the original isovalue F_g^2. The affine approximation of the reconstruction kernel in image space is thus

$$r'_k(\mathbf{x}) = \frac{1}{|\mathbf{Q}'''|^{1/2}} g_{\mathbf{Q}'''^{-1}}(\mathbf{x} - \mathbf{x}_t),$$

where $\mathbf{Q}'''$ is obtained by scaling Equation 10 to match the isovalue F_g^2, i.e.:

$$\mathbf{Q}''' = \frac{F_g^2}{f - dx_t - ey_t}\mathbf{Q}''. \tag{11}$$

Since we choose the isovalue F_g^2 to correspond with the cutoff value of the kernels, the *shape* of the truncated kernels is correct under projective mappings. This is illustrated in Figure 2, where we compare our novel approximation with the previous approximation used by Heckbert [7] and Zwicker et al. [24]. In their techniques, the affine approximation is correct at the center of the kernel, i.e., the mappings of the kernel *centers* correspond. In contrast, our technique is correct for a conic *isocontour*. As illustrated in the bottom row of Figure 2, the previous affine approximation can lead to serious artifacts for extreme perspective projections. Figure 3 further illustrates the difference between our novel approximation and previous techniques. Here we rendered a point-sampled plane with reconstruction kernels that are truncated such that they exactly touch each other in 3D. Since our approximation of the perspective projection is exact at the cutoff value of the kernels, the projected kernels touch each other also in the image plane. On the other hand, the previous affine approximation clearly leads to splat shapes that are not perspective correct.

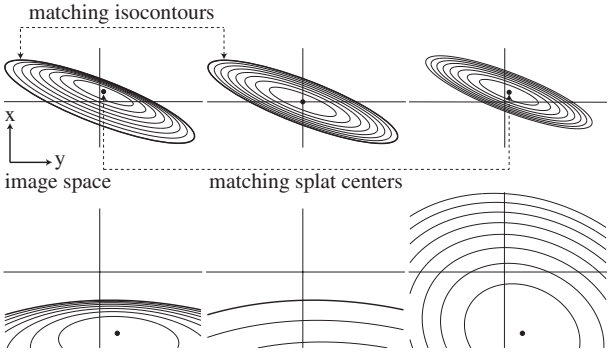

Figure 2: Left: *Projectively mapped isocontours of a Gaussian.* Middle: *Our affine approximation; the outermost isocontour is correct.* Right: *Heckbert's affine approximation; the center is correct.* Bottom row: *A particularly bad case for the previous approximation.*

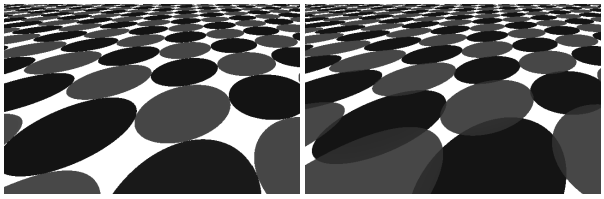

*Figure 3: Perspective projection of a point-sampled plane. Our approximation (*left*) leads to perspective correct splat shapes, in contrast to the previous approximation (*right*).*

5 Rendering Sharp Features

For many applications, the ability to render surfaces with sharp features, such as corners or edges, is a requirement. From a signal processing point of view, these features contain infinite frequencies. Hence, to convert them into an accurate discrete representation, the sampling rate should be infinitely high. In other words, we would need to store a large number of very small reconstruction kernels to capture the unbounded frequencies of the features. Since this approach is not practical, we instead include an *explicit representation* of sharp features in our surface

description, similar as proposed by Pauly [20]. Sharp features can either be extracted from point data using automatic feature detection algorithms [19], or they may be generated explicitly during the modeling process, for example by performing CSG operations [20].

We represent a feature by storing so called *clip lines* defined in the local tangent planes of the reconstruction kernels. As illustrated in Figure 4, clip lines are computed by intersecting the tangent planes of adjacent reconstruction kernels on either side of a feature, hence providing linear approximations of features. Clip lines divide the tangent planes into two parts: in one part, the reconstruction kernel is evaluated as usual, while in the other it is discarded, or clipped. Since pairs of reconstruction kernels share the same clip line, no holes will appear due to clipping. Rendering a clipped reconstruction kernel

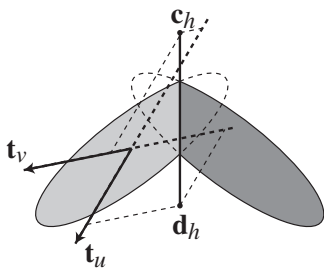

Figure 4: Clip line defined by points $\mathbf{c}_h$ and $\mathbf{d}_h$ on the intersection of two tangent planes.

for a point $\mathbf{p}_k$ is straightforward. A clip line is represented using two homogeneous points $\mathbf{c}_h$ and $\mathbf{d}_h$ in its local tangent plane. These points are projected to image space by multiplying them with the projective mapping matrix, yielding $\mathbf{c}'_h = \mathbf{c}_h\mathbf{M}_k$ and $\mathbf{d}'_h = \mathbf{d}_h\mathbf{M}_k$. We then perform projective normalization yielding non-homogeneous points $\mathbf{c}'$ and $\mathbf{d}'$, and we compute a direction vector $\mathbf{v}$ that is perpendicular to the line through these points. We evaluate the reconstruction kernel at a point $\mathbf{x}$ in image space if $(\mathbf{x} - \mathbf{c}') \cdot \mathbf{v} > 0$, otherwise we discard $\mathbf{x}$.

To illustrate this technique we applied a color texture to a point-sampled cube, shown in Figure 5. Note that the color texture is reconstructed smoothly on the faces of the cube (Figure 5 left). On the other hand, splat clipping allows the accurate rendering of sharp edges and corners without blurring or geometric artifacts. We represent edges by a single clip line per splat, while corners require two clip lines. In the close-up on the right in Figure 5, we used a cutoff value of $F_g = 0.5$ to further emphasize the effect of splat clipping.

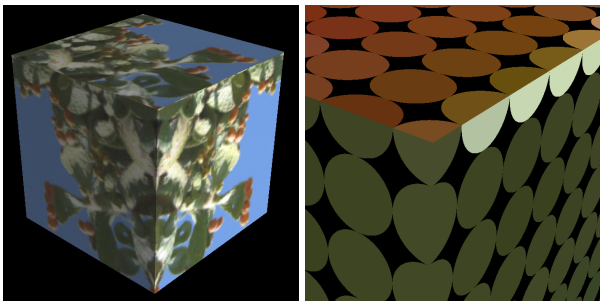

Figure 5: Rendering with clipped splats.

6 Implementation

We have implemented a point rendering algorithm based on our novel approximation of the projective mapping using vertex and fragment programs of modern GPUs. The algorithm proceeds in three passes very much like previous methods for hardware accelerated point splatting, described in more detail in [1, 6, 22]. In the first pass, we render a depth image of the scene that is slightly offset along the viewing rays. In the second pass, splats are drawn using only depth tests, but no updates, and color blending enabled. Hence color values are accumulated in the framebuffer, effectively computing the weighted sum in Equation 2. Similar as in [1, 6], splats are rendered using OpenGL points instead of quads or triangles. The final pass performs normalization of the color values by dividing through the accumulated filter weights.

However, our technique differs from [1, 6, 22] in the way splats are computed and rasterized. In the vertex program we compute the variance matrix $\mathbf{R}'_k + \mathbf{H}$ of the resampling filter (Equation 3). Here, we use our novel approximation $\mathbf{Q}'''$ (Equation 11) for the variance matrix of the reconstruction kernel $\mathbf{R}'_k$. Note that computing $\mathbf{Q}'''$ requires the inversion of the projective mapping $\mathbf{M}_k$, which hovewer might be numerically ill-conditioned, e.g., if the splat is about to degenerate into a line segment. We detect this case by checking the condition number of $\mathbf{M}_k$, and we discard the splat if it is above a threshold. This is similar to the approach proposed by Olano and Greer for triangle rendering [16]. Also note that $\mathbf{Q}'''$ represents a general conic (not necessarily an ellipse) because of the projective mapping that has been applied. We use the criteria given in Section 4.2 to determine if the conic is an ellipse and discard the splat otherwise. Finally, the vertex shader also determines the OpenGL point size by computing a bounding box using the method described in Section 4.2.

The 2×2 conic matrix $(\mathbf{Q}''' + \mathbf{H})^{-1}$ of the resampling filter is then passed to the fragment program. The frag-

ment program is executed at every pixel $\mathbf{x}$ covered by the OpenGL point primitive, evaluating the ellipse equation $r^2 = \mathbf{x}(\mathbf{Q}''' + \mathbf{H})^{-1}\mathbf{x}^T$. If the pixel is inside the ellipse, i.e., $r^2 < F_g^2$, r^2 is used as an index into a lookup-table (i.e., a 1D texture) storing the Gaussian filter weights. Otherwise the fragment is discarded. Note that $\mathbf{x}$ are pixel coordinates, which are available in the fragment program through the `wpos` input parameter [13]. Hence the fragment program does not rely on point sprites. For the normalization pass the result of the second pass is copied to a texture. Rendering one window-sized rectangle with this texture sends all pixels through the pipeline again, hence a fragment program can do the required division by alpha.

We have developed two implementations of our algorithm, an implementation using Cg [13] and a hand-tuned assembly code version, both providing the same functionality. Average timings over a number of objects containing 100k to 650k points for a PC system with a GeForceFX 5950 GPU and a Pentium IV 3.0 GHz CPU are reported in Table 1. We measured performance of single pass rendering (simple z-buffering, without splat blending and normalization) and the three pass algorithm described above. The numbers of vertex and fragment program instructions without splat clipping are summarized in Table 2. Note that the test objects do not contain any clipped splats, which require a few additional instructions. However, overall rendering performance is not affected significantly by splat clipping. Typically only few splats require clipping and we switch between two different fragment shaders to rasterize clipped and non-clipped splats to avoid overhead.

	512×512		1280×1024	
	1 pass	3 pass	1 pass	3 pass
Cg	2.9	1.4	2.8	1.2
Assembler	11.2	3.1	10.4	2.8

Table 1: Rendering timings for window resolutions of 512×512 and 1280×1024 in million splats per second.

	Cg		Assembler	
	VP	FP	VP	FP
Pass 1	151	28	109	9
Pass 2	164	22	120	13
Pass 3	-	4	-	3
Single pass	149	23	108	7

Table 2: Number of vertex (VP) and fragment program (FP) instructions for Cg and assembler implementations.

7 Results

Our approach implements an EWA filter as proposed by Heckbert [7], including a Gaussian prefilter. The EWA filter is an anisotropic texture filter providing high image quality avoiding most aliasing artifacts as illustrated at the top in Figure 6. In contrast, splatting without the prefilter leads to Moiré patterns, shown at the bottom in Figure 6. Our new affine approximation to the projective mapping results in perspective correct splat shapes. Previous approaches lead to holes in the rendered image if the model is viewed from a flat angle and additionally shifted sufficiently from the viewing axis. These artifacts are effectively avoided by our method, as shown in Figure 7. Although our method does not map the splat centers to the perspective correct position (Section 4.4) we have not observed any visual artifacts due to this approximation. In Figure 8 we illustrate the rendering of objects with sharp features. This geometry, which has been created using CSG operations, consists of only 6433 splats. Because of splat clipping, the sharp edges can still be represented and displayed quite accurately, as is shown in the close-up on the right.

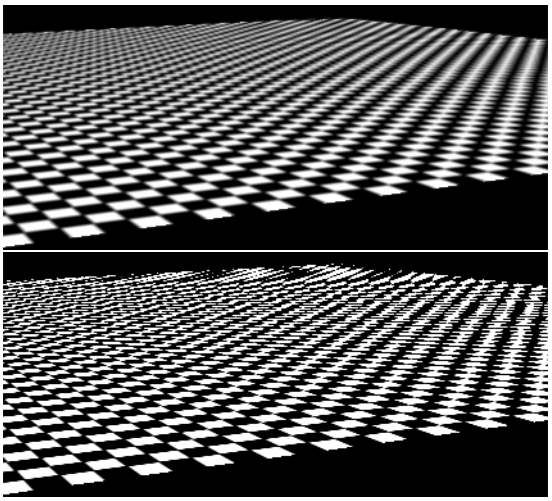

Figure 6: Checkerboard texture rendered with perspective accurate splats: (top) with and (bottom) without prefiltering.

8 Conclusions

We have presented a novel approach for accurate, high quality EWA point splatting, introducing an alternative approach to approximating the perspective projection of the Gaussian reconstruction kernels. This approximation is based on the exact projective mapping of conic sections, which we derived using homogeneous coordi-

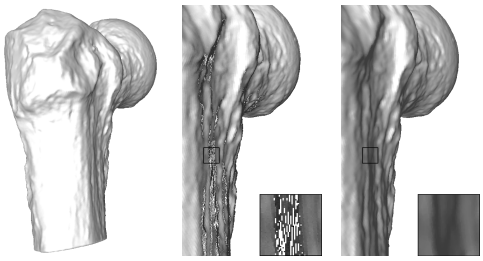

*Figure 7: Previous affine approximations lead to holes in the reconstruction for extreme viewing positions (*center*). Our novel approximation avoids these artifacts (*right*).*

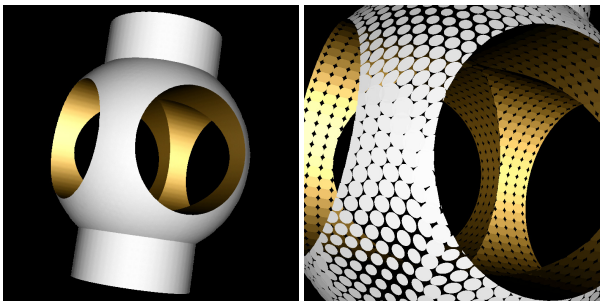

Figure 8: Rendering an object with sharp features created by CSG operations.

nates. It leads to perspective accurate splat shapes in image space, hence avoiding artifacts of previous approaches. Further, we extended the splat primitive by adding clip lines, which allows us to represent and render sharp edges and corners. We described a rendering algorithm for those primitives that can be implemented entirely on modern GPUs. While the parameters of the display primitive are computed in the vertex stage, rasterization is performed in the fragment programs using point primitives.

From a more general point of view, we have illustrated that the programmability of modern GPUs allows the efficient use of other rendering primitives than triangles, completely replacing the built-in primitive setup and rasterization stages. This approach could be exploited to render primitives such as polygons with more than three vertices, or combinations of polygons and conics, etc. It would also be interesting to include curvature information, similar as proposed in [8, 9]. Another direction for future work is the integration of our approach with efficient, hierarchical data structures such as sequential point trees [3], which allow level-of-detail rendering completely on the GPU.

References

[1] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern GPUs. In *Pacific Graphics 2003*, pages 335–442, 2003.

[2] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Eurographics Workshop on Rendering*, pages 53–64, 2002.

[3] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. In *SIGGRAPH 2003*, pages 657–662, 2003.

[4] N. Greene and P. Heckbert. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE CG & A*, 6(6):21–27, 1986.

[5] J. P. Grossman and W. J. Dally. Point sample rendering. In *Rendering Techniques '98*, pages 181–192, July 1998.

[6] Gael Guennebaud and Mathias Paulin. Efficient screen space approach for hardware accelerated surfel rendering. In *Vision, Modeling and Visualization, Munich*, pages 1–10, November 2003.

[7] P. Heckbert. Fundamentals of texture mapping and image warping. M.sc. thesis, University of California, Berkeley, June 1989.

[8] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In *Rendering Techniques '01*. Springer Verlag, 2001.

[9] Aravind Kalaiah and Amitabh Varshney. Modeling and rendering points with local geometry. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):30–42, January 2003.

[10] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. In *SIGGRAPH 2000*, pages 131–144, 2000.

[11] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, Univ. of North Carolina at Chapel Hill, 1985.

[12] E. Lindholm, M. Kilgard, and H. Moreton. A user-programmable vertex engine. In *SIGGRAPH 2001*, pages 149–158, 2001.

[13] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: A system for programming graphics hardware in a C-like language. In *SIGGRAPH 2003*, 2003.

[14] W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3d photography using opacity hulls. In *SIGGRAPH 2002*, pages 427–437, 2002.

[15] J. McCormack, R. Perry, K. Farkas, and N. Jouppi. Feline: fast elliptical lines for anisotropic texture mapping. In *SIGGRAPH 1999*, pages 243–250, 1999.

[16] Marc Olano and Trey Greer. Triangle scan conversion using 2d homogeneous coordinates. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware 1997*, pages 89–96, 1997.

[17] M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In *SIGGRAPH 2001*, pages 379–386, 2001.

[18] M. Pauly, M. Gross, and L. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Vis. 2002*, pages 163–170, 2002.

[19] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces. In *Eurographics 2003*, pages 281–289.

[20] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In *SIGGRAPH 2003*, pages 641–650, 2003.

[21] H. Pfister, M. Zwicker, J. VanBaar, and M. Gross. Surfels: Surface elements as rendering primitives. In *SIGGRAPH 2000*, pages 335–342, 2000.

[22] L. Ren, H. Pfister, and M. Zwicker. Object space ewa surface splatting: a hardware accelerated approach to high quality point rendering. In *Eurographics 2002*, pages 461–470, 2002.

[23] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH 2000*, pages 343–352, 2000.

[24] M. Zwicker, H. Pfister, J. VanBaar, and M. Gross. Surface splatting. In *SIGGRAPH 2001*, pages 371–378, 2001.