Eurographics Symposium on Point-Based Graphics (2006)
M. Botsch, B. Chen (Editors)

Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering

Gaël Guennebaud, Loïc Barthe and Mathias Paulin†

IRIT - CNRS - Université Paul Sabatier - Toulouse - France

Abstract

In this paper we present multiple simple and efficient improvements for splatting based rendering systems. In a first step we derive a perspectively correct splat rasterization algorithm suitable for both efficient implementation on current GPU and the design of fast dedicated rasterization units taking advantages of incremental calculations. Next, we propose a new efficient and high-quality approximation of the optimal EWA (Elliptical Weighted Average) filtering framework. We also show how transparent point-clouds can be rendered by current GPU in an order independent way by using a modified depth-peeling approach. Finally, in the context of hybrid points and polygons rendering systems, we present a simple mechanism to smooth the transitions between the different representations. Combined to previous contributions in the field, these improvements lead to a high-quality, performant and full-featured hardware-oriented splatting rendering system.

1. Introduction

Owing to their high flexibility, point-based surface representations have been successfully used in a wide range of applications. Indeed, from interactive surface modeling [ZPKG02, PKKG03] to real-time visualization of highly complex-scenes [WFP*01, GBP04], points have exhibited several advantages against classical polygons.

The common denominator of every graphic application is the visualization stage. According to the end purpose of the application, radically different approaches can be used to render a point cloud. When very high-quality is required, a ray tracing approach (backward projection) is probably the best choice. Most ray tracing methods are based on a local moving least square (or variants) surface reconstruction [Lev03] allowing to accurately intersect the point cloud [AA03, AKP*05]. One of the most interesting aspect of ray tracing approaches is that they allow the implementation of realistic shading calculation [SJ00]. However, in spite of several optimizations [WS05], such approaches remain too expensive for interactive purpose.

On the other hand, rasterization approaches (forward projection) usually offer a significantly faster rendering and a higher degree of flexibility since, under a given geometry complexity threshold, the geometry can be directly rendered without the need of any special data structure. Such an approach is usually based on a splat representation (a set of oriented disks or ellipses) and is implemented with the so called *splatting* algorithm. Initially, high-quality splatting can only be obtained via a pure software implementation [ZPvG01]. Fortunately, recent GPU capabilities now allow us to implement efficient splatting algorithms on the GPU with the same rendering quality as software implementations [BHZK05]. To provide high-quality, the key features of a splatting algorithm are:

1. a perspective correct splat rasterization avoiding holes,
2. a band limiting pre-filter for high-quality texture filtering (EWA filtering),
3. a deferred shading approach smoothly blending the splat attributes before the shading calculations.

The last criterion is commonly referred as per-pixel shading. One of the most advanced hardware-accelerated splatting implementation is probably the one of Botsch et al. [BHZK05] since it fully satisfies the above 1 and 3 criteria. We however see two drawbacks in their approach that we overcome in this paper. First, from the rendering quality point of view, their coarse approximation of the EWA filtering may fail to completely remove the aliasing in case of both minification and magnification. Secondly, from the performance point of view, both their splat rasterization method and EWA filtering approximation require quite expensive computations at the fragment level without offering any pos-

† e-mail: {guenneba | lbarthe | paulin}@irit.fr

sibility to develop an optimized rasterization unit taking advantage of incremental calculations. Moreover, none of GPU based splatting techniques have yet provided a mechanism to support transparent models.

Beyond purely point-based rendering systems, when very high performance matters more than flexibility, several works have already shown that mixing splats with polygons allows to reach outstanding performances [CAZ01, CN01, WFP*01, CH02, DVS03]. Indeed, even though points are particularly efficient to handle and render complex geometries, they are also significantly less efficient to represent flat surfaces. Because the *flatness* of the surface is relative to the current view point, such an hybrid rendering system dynamically selects for each part of the model the best representation at the best level-of-details. However, none of these works have already addressed the problem of the discontinuities which may occur at the transition between polygonal and splat representations.

Then, the purpose of this paper is to present some simple but useful splatting improvements allowing us to overcome the limitations of existing splatting rendering systems pointed out above. Especially, the contributions of this paper include:

- an efficient perspective correct splat rasterization algorithm suitable for fast incremental calculations as well as fast GPU implementation (section 3),
- a high-quality and efficient approximation of the EWA resampling filter (section 4),
- a high-quality hybrid splat-polygon rendering system with smooth transitions (section 5),
- a GPU accelerated order independent splatting algorithm of transparent models (section 6).

The improvements proposed in this paper are simple, practical and very useful to provide high-quality rendering of point based geometries in real-time.

2. Related Work

This section discusses recent splatting based rendering techniques. More detailed surveys on point-based techniques and point-based rendering can be found in [KB04, SP04].

A rigorous splatting framefork has been first introduced by Zwicker et al. [ZPvG01] in 2001. This approach uses a deferred shading strategy where the surface attributes are reconstructed before the shading calculations, thus leading to high-quality per-pixel shading effects. The surface attributes are reconstructed in the screen space by accumulating the weighted attributes held by each splat. During this resampling step, aliasing artifacts are avoided by EWA filtering: high frequencies are removed by applying a screen space low-pass filter before the splat rasterization. Owing to the overlapping of splats, visibility computations are performed by a fuzzy z-buffer which compares the depth values with a small tolerance threshold ε such that overlapping fragments of the same surface are effectively blended. The use of a modified A-buffer allows them to render transparent models in a single pass and in an order independent way. Although their approach provides high visual quality, this renderer requires an expensive splat rasterization setup and it is purely software based. Therefore its performances are limited to about 1M splats per second.

Most of further researches try to overcome these last limitations via hardware-accelerated splatting approaches built on some restrictions, approximations, or variants of the original surface splatting. The first limitation for a GPU implemention is the lack of a fuzzy z-buffer. This limitation is overcome via a multipass algorithm [RPZ02] shared by all GPU based splatting approaches. In the first *visibility splatting* pass, splats are rasterized in order to precompute a hole free depth buffer. The depth values of splats are shifted by ε along the view direction. In the second pass, that we further refer as the *attribute accumulation* pass, splats are rendered with lighting and additive blending. Owing to the previous pass, only effectively visible contributions are accumulated in the destination buffer. In the final *normalization* pass, each pixel of the resulting buffer is normalized by the sum of weights stored in the alpha channel.

Although some methods have used polygonal primitives to render splats [KV01, RPZ02, PSG04], the pixel shader capabilities of GPUs now allow us to efficiently draw a splat via the hardware point primitive [GP03, BK03]. Such an approach significantly improves the rendering performance and flexibility, since only one vertex per splat is sent to the GPU, instead of 3 or 4 vertices for the previous methods. Since the hardware point primitive only allows to generate fragments under a custom axis aligned screen space square area, the pixel weight and depth values must be computed by a custom fragment shader. However, in all these methods, the perspective projection is always approximated by an affine mapping and hence, small holes may appear. To overcome this problem, Zwicker et al. [ZRB*04] propose an affine approximation which accurately maps the splat contours while in [BSK04, BHZK05] Botsch et al. use a ray casting approach. Sharp features can be represented and rendered using splats clipped by lines defined in their local tangent frame [PKKG03, ZRB*04].

The first GPU based methods, providing per-pixel lighting, store for each splat a precomputed normal field which is used to derive per-pixel normals [KV01, BSK04]. However, due to the need of preprocess, such approaches have a low degree of flexibility and are also rather expensive since the shading computations are uselessly performed on a large number of pixels. Since almost two years, GPU provides all the required features to implement a splatting algorithm with deferred shading capabilities [BHZK05]. In such a configuration, during the second splatting pass, instead of the shaded color, the splat attributes required by the shading calcula-

tions (e.g. color, normal and depth value) are accumulated into attribute buffers. During the final normalization pass, attributes are normalized by the sum of weights and these per-pixel reconstructed values are used to compute the final output color. In addition to increase the visual quality, a deferred shading approach also allows to implement complex shading algorithm with a very low overhead since the shading calculations are performed on visible pixels only.

The splatting algorithms proposed in this paper are built in such a multipass with deferred shading fashion.

3. Perspective Correct Splatting

In this section we explain how to perform a perspective correct rasterization of a general elliptic splat. Compared with the raycasting approach of Botsch et al. [BSK04, BHZK05], the approach described here is significantly faster and is suitable for a fast incremental implementation. This last point is especially important in order to develop dedicated hardware or software splat rasterization units. Our approach relies on standard perspective correct polygon rasterization techniques.

In a first step, let us remind the general splatting process. Let S_k be a splat defined by its center $\mathbf{p}_k = (p_x, p_y, p_z)^T$ and a local tangent frame ($\mathbf{s}_k$, $\mathbf{t}_k$) where the two tangent vectors $\mathbf{s}_k = (s_x, s_y, s_z)^T$ and $\mathbf{t}_k = (t_x, t_y, t_z)^T$ are scaled according to the principal radii of the elliptical splat. In this local frame, a point $\mathbf{q} = (u, v)^T$ is inside the splat if and only if $u^2 + v^2 \leq 1$. In order to continuously reconstruct the surface attributes, a 2D reconstruction kernel $r_k(\mathbf{q})$ defined in the tangent plane is associated to each splat. A typical choice for r_k is a radially symmetric Gaussian. Then, the rendering process is achieved by projectively mapping the reconstruction kernels from their local tangent frames to the image plane. At each image sample $\mathbf{x}$, each contribution is accumulated and normalized by the sum of weights. For a given scalar attribute f_k, its reconstructed value $f'(\mathbf{x})$ at the location $\mathbf{x}$ is then:

$$f'(\mathbf{x}) = \frac{\sum_k f_k r_k(\mathcal{M}_k^{-1}(\mathbf{x}))}{\sum_k r_k(\mathcal{M}_k^{-1}(\mathbf{x}))} = \frac{\sum_k f_k r'_k(\mathbf{x})}{\sum_k r'_k(\mathbf{x})} \tag{1}$$

where $\mathcal{M}_k$ is the 2D-2D projective mapping from the local tangent frame of the splat S_k to the image space. Also, $r'_k(\mathbf{x})$ denotes the warped reconstruction kernel in the image space.

3.1. Perspective Splat Rasterization

Now, in order to simplify the following equations, we define the 2D image plane by its center $(0,0,1)^T$ and tangent vectors $(1,0,0)^T$ and $(0,1,0)^T$. The projective mapping $\mathcal{M}_k(\mathbf{q})$ of a point $\mathbf{q} = (u, v)^T$ onto the image plane through the origin $(0,0,0)^T$ is then defined as follow:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathcal{M}_k(\mathbf{q}) = \begin{bmatrix} \frac{us_x+vt_x+p_x}{us_z+vt_z+p_z} \\ \frac{us_y+vt_y+p_y}{us_z+vt_z+p_z} \end{bmatrix} \tag{2}$$

Using homogeneous coordinates, this mapping can be expressed as the product of the 3×3 matrix $\mathbf{M}_k$ with the homogeneous point $\mathbf{q} = (u, v, 1)^T$:

$$\begin{bmatrix} xz \\ yz \\ z \end{bmatrix} = \mathbf{M}_k \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{s}_k & \mathbf{t}_k & \mathbf{p}_k \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3}$$

During the rasterization process we need to inverse this mapping (equation 1). The inverse of a projective mapping is also a projective mapping and hence we have just to inverse the matrix $\mathbf{M}_k$. In practice any scalar multiple matrix $\mathbf{N}_k = \alpha\mathbf{M}^{-1}$ with $\alpha \in \mathsf{R}^*$ can be used instead of the exact inverse matrix $\mathbf{M}_k^{-1}$ to define the inverse mapping $\mathcal{M}_k^{-1}$:

$$\begin{bmatrix} uw \\ vw \\ w \end{bmatrix} = \mathbf{N}_k \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4}$$

For instance we can use for $\mathbf{N}_k$ the adjoint $adj(\mathbf{M}_k)$ which is efficiently computed by only three cross products (a cross product is a native instruction on current GPU):

$$adj(\mathbf{M}_k) = \begin{bmatrix} \mathbf{t}_k \times \mathbf{p}_k \\ \mathbf{p}_k \times \mathbf{s}_k \\ \mathbf{s}_k \times \mathbf{t}_k \end{bmatrix} \tag{5}$$

Moreover, in order to perform the visibility computation, we also need to compute the window space depth value $d(\mathbf{x})$ of the current splat at every image location $\mathbf{x}$. Most of z-buffer algorithm does not directly use the view space z value as the window space depth value but rather $1/z$ linearly mapped to $[0,1]$, i.e. $d(\mathbf{x}) = a + b/z$ where a and b are constants defined by the near and far plane of the view frustum. Since $d(\mathbf{x})$ is a linear transformation of $1/z$, it can be linearly interpolated in the image space. Again, since we can use any scalar multiple (not null) of the matrix $\mathbf{M}_k^{-1}$ to compute the inverse mapping, an optimization is to use for the matrix $\mathbf{N}_k$ the matrix $adj(\mathbf{M}_k)$ scaled such that the homogeneous coefficient w in the equation 4 is equal to b/p_z. If $adj(\mathbf{M}_k)_3$ denotes the third row of the matrix $adj(\mathbf{M}_k)$, then the matrix $\mathbf{N}_k$ is:

$$\mathbf{N}_k = \frac{b}{adj(\mathbf{M}_k)_3 \cdot \mathbf{p}_k} adj(\mathbf{M}_k) \tag{6}$$

Then, the computation of $d(\mathbf{x})$ requires only a single addition per pixel $\mathbf{x}$. In practice, because both current GPU and CPU have four component vector instructions, the product of a 2D homogeneous vector with a 4×3 matrix is not more expensive than the one with a 3×3 matrix (if we assume that the matrix is stored in three vector registers of four components). Therefore, a finer optimization is to use the fourth row to directly compute the final depth value. The value of the fourth row $(\mathbf{N}_k)_4$ is the third row of the matrix $adj(\mathbf{M}_k)$

scaled as above but with the offset a added to the third components:

$$(\mathbf{N}_k)_4 = \frac{b}{adj(\mathbf{M}_k)_3 \cdot \mathbf{p}_k} adj(\mathbf{M}_k)_3 + \begin{bmatrix} 0 \\ 0 \\ a \end{bmatrix} \qquad (7)$$

Finally, because the three coordinates (uw, vw, w) (or four if we use the second optimization) can be linearly interpolated in the image space, it is easy to derive a very efficient dedicated splat rasterization unit taking advantage of incremental calculations.

3.2. GPU Implementation details

In practice, since we cannot modify the rasterization unit of current GPU, as several other GPU-based splatting implementation [GP03, BK03, ZRB*04, BSK04, BHZK05] we use a vertex shader to simulate the splat rasterization *setup*, the hardware point primitive to generate fragments and a fragment shader to both evaluate the reconstruction kernel and compute an accurate depth value.

Splats are sent to the GPU using the standard point primitive. The vertex shader, in addition to common space transformations, computes the matrix $\mathbf{N}_k$ defining the inverse mapping $\mathcal{M}_k^{-1}$ as well as a conservative 2D axis aligned bounding box of the projected splat. The position and size of the point primitive are respectively the center and the largest dimension of the box. The fragment shader required by our approach is very simple, basically only three computational instructions, and hence it is significantly more efficient than previous approaches. Indeed, if we transmit the above matrix to the fragment shader via three column vectors, the matrix multiplication requires only two `MAD` instructions (the `MAD` instruction of GPU performs one multiplication followed by one addition). Next, the 2D kernel is precomputed in a 2D texture such that the weight $r'(\mathbf{x})$ is obtained by a single projective texture fetch. We remind that the accurate window space depth value is simply the fourth component coming from the previous matrix product. Finally, fragments outside the splats are removed by the alpha test unit, while the multiplication of the attributes by the weight and their accumulation are performed by the blending unit.

4. High Quality Filtering

In the previous section we have directly rasterized the object space reconstruction filter r_k without taking into account the pixel sampling grid. Thus, in the case of minification and high frequencies, resampling filters may miss pixels leading to aliasing artifacts (figure 3-top). In order to overcome this problem, in the original EWA splatting framework [ZPvG01], reconstruction kernels are band limited by convolution with a screen-space pre-filter. This leads to a so called *resampling filter* which is effectively rasterized. A convolution approach is only conceivable if the result is simple and can be determined analytically. This is why in the EWA framework, an affine mapping approximation is used instead of the projective mapping and Gaussians are used for both the reconstruction kernel and the low pass filter.

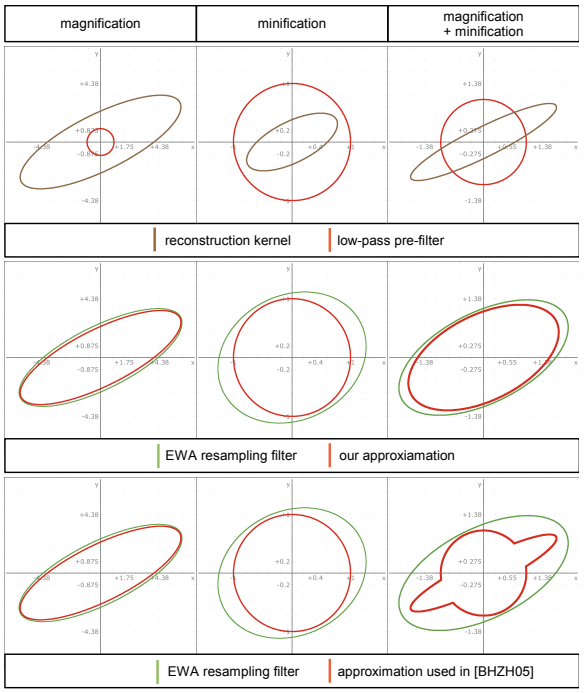


Figure 1: *Comparison of the shapes of our EWA approximation to the rigorous EWA resampling filter and the approximation of Botsch et al. [BHZK05].*

Unfortunately, the projective mapping of a Gaussian is not a Gaussian, and hence it is not possible to derive a resampling filter by convolution and some approximations have to be done. For instance, in [BHZK05] the screen-space pre-filter is applied by taking the maximal value of the reconstruction kernel and the low-pass filter results. However, this approximation has two major drawbacks. From the performance point of view, it requires to evaluate, for each fragment, both the resampling kernel and the low-pass filter, that is quite expensive since the fragment stage is usually the bottleneck of splatting systems. From the quality point of view, this approximation is very coarse as soon as the splat is in the case of minification in one direction and magnification in the other direction (figure 1). In such cases, fine parts of the splat may still fall between image samples.

4.1. Our EWA approximation

We propose a new approximation being in between the optimal EWA resampling filter and the approximation of [BHZK05]. The purpose of our approximation is to dynamically adjust the tangent vectors of the splat in the object space so that the pre-filter can always be included in the perspectively mapped reconstruction filter r'_k. In this case,

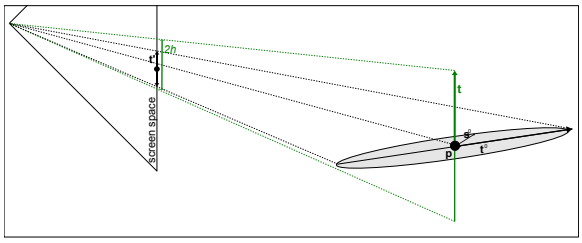


Figure 2: *Illustration of the principle of our EWA filter approximation. The diameter edge of the splat along the tangent vector* $\mathbf{t}^0$ *is projected onto the screen to obtain* $\mathbf{t}'$*. Since its length is lower than the pre-filter radius h, the tangent vector* $\mathbf{t}$ *is* $\mathbf{t}^0$ *adjusted such that it is parallel to the screen plane and its projection size is equal to the pre-filter radius.*

the warped reconstruction filter will still have the shape of an ellipse and will never fall between samples of the output image. In practice, a such low-pass filtering approach only holds if it is computationally significantly less expensive than previous methods. Moreover, our low-pass filter is only applied in case of minification, i.e. when the reconstruction kernel has a screen space size lower than a few pixels. Thus, it is pertinent to use approximations in order to get the maximal performance, and we propose to simply adjust the tangent vectors in an independent manner.

Let h be the radius of the low pass filter in the normalized image plane for which a typical choice is $\sqrt{2}\frac{2}{\mathrm{vp}_w}$ (here vp_w denotes the viewport width). Let us denote the initial tangent vectors of a given splat as $\mathbf{s}_k^0$ and $\mathbf{t}_k^0$. The final vectors $\mathbf{s}_k$ and $\mathbf{t}_k$ are computed identically. The vector $\mathbf{t}_k$ is obtained by adjusting the vector $\mathbf{t}_k^0 = (t_x, t_y, t_z)^T$ such that the length of the half of the perspective projection $\mathbf{t}_k'$ of the diameter edge $(\mathbf{p}_k - \mathbf{t}_k^0, \mathbf{p}_k + \mathbf{t}_k^0)$ onto the screen space is greater than the radius of the low-pass filter (figure 2). The projected vector $\mathbf{t}_k'$ is given by:

$$\mathbf{t}_k' = \frac{1}{p_z^2 - t_z^2} \begin{bmatrix} t_x p_z - t_z p_x \\ t_y p_z - t_z p_y \\ 0 \end{bmatrix} \tag{8}$$

If $\|\mathbf{t}_k'\|$ is already greater than the radius h, then $\mathbf{t}_k$ is directly $\mathbf{t}_k^0$. Otherwise, we take $\mathbf{t}_k$ as the projection of $\mathbf{t}_k^0$ onto the image plane scaled in order to satisfy our criterion:

$$\mathbf{t}_k = \frac{\mathbf{t}_k'}{\|\mathbf{t}_k'\|} p_z h \tag{9}$$

Finally, the reconstruction kernel is rasterized as explain in the previous section.

However, independently adjusting the tangent vectors as explain above only holds if their respective projections onto the screen space are in orthogonal direction and thus aligned with the principal axis of the warped reconstruction kernel. Even though this assumption cannot be satisfied in general, we propose a simple heuristic computing the initial tangent vectors $\mathbf{s}_k^0$ and $\mathbf{t}_k^0$ while providing very good results. Indeed, for an isotropic splat of normal $\mathbf{n}_k$, any orthogonal tangent vectors can be used to define the local parametrization and our heuristic holds in the two following cross products:

$$\begin{aligned} \mathbf{s}_k^0 &= \mathbf{p}_k \times \mathbf{n}_k \\ \mathbf{t}_k^0 &= \mathbf{n}_k \times \mathbf{s}_k^0 \end{aligned} \tag{10}$$

We can characterize the accuracy of the basis vectors by the minimal angle value α formed by their screen space projections. Then, for a view frustum with an aperture angle of 90 degrees, this heuristic generates basis vectors with an α value between 90 and 60 degrees and with an average of 81.7 degrees. When the α value is not equal to 90 degrees, the screen-space low-pass filter could be slightly truncated. Indeed, in the worst case ($\alpha = 60$), our algorithm may *forget* to adjust the reconstruction filters for which the minimal screen space width is $h\sqrt{2}$ (our threshold being $2h$), that, in fact, does not matter since it is still greater than the output resolution.

We point out that unlike the previous approaches, the depth correction is lost for splats which are in the case of minification because the tangent vectors become parallel to the screen plane. This comportment is intentional because it prevents from some visibility artifacts that previous approaches may exhibit in some particular cases. Indeed, if we do not transform the tangent plane of a minificated splat, then this splat can lead to arbitrary depth value since the required stretch value is potentially infinite. Moreover, in practice, the depth correction is not a requirement for splats which are below a single pixel.

4.2. Comparison

On the one hand, the main limitation of our approach is that it works well for isotropic splats only. On the other hand, excepted when external anisotropic forces are applied to the point set [PKKG03, MKN*04], isotropic splats are more widely used than elliptic splats because they do not require any expensive preprocessing. Moreover, our high frequencies filtering approach exhibits the following nice features:

- Splats are extended only in the directions of minification: unlike the original EWA filtering approach, no additional fragments are generated when it is not necessary.
- All computations are done during the splat setup: unlike the previous EWA approximation [BHZK05] there is no overhead at the rasterization step.
- In case of both minification and magnification, our approximation is closer to the original EWA resampling filter than the previous approximation of Botsch et al. [BHZK05] (see figures 1 and 4).

5. High-Quality Hybrid Rendering

In this section we show how a splatting rendering system can be easily extended to handle hybrid spats and polygons high-quality rendering.

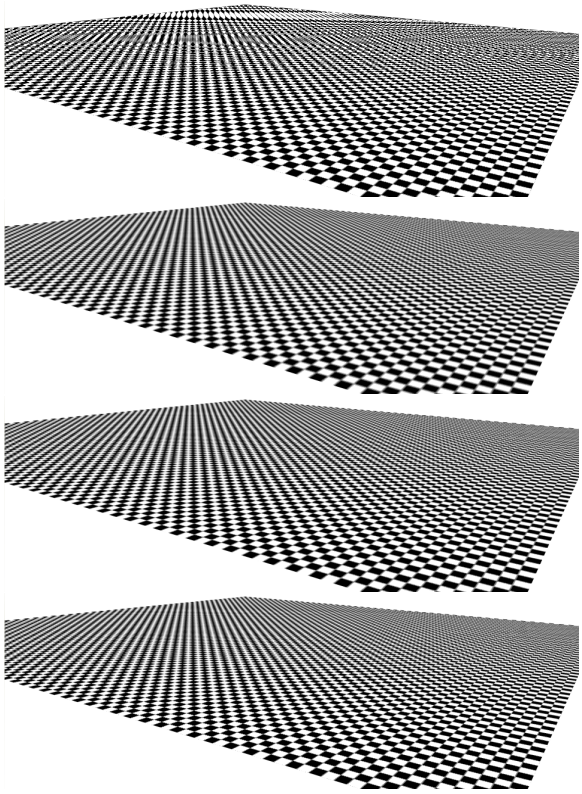


Figure 3: *A 600×600 points checkerboard rendered from top to bottom: without screen space pre-filter, with EWA filtering, with our EWA approximation and with the EWA approximation of Botsch et al. [BHZK05].*

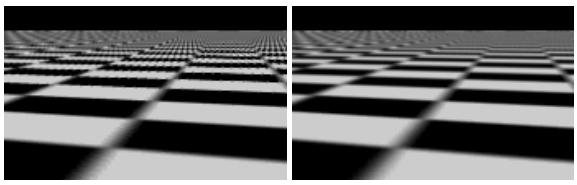


Figure 4: *A close view of a checkerboard rendered from left to right with the EWA approximation of Botsch et al. [BHZK05] and with our EWA approximation.*

5.1. Transitions smoothing

Whereas several results have already proven the efficiency of such hybrid rendering systems, none of them focus on the problem of the transitions between the polygonal and the splat representations. Indeed, when a surface is rendered via both splats and polygons, if no special treatment is performed, discontinuities appear at the transition level (figure 5a). It is hence essential to provide a mechanism to smooth these transitions. To do so, two properties are required: first, splats and polygons must overlap (that is almost always the case) and secondly, we need for every pixels a weight value continuously varying from 0 to 1 in the overlapping regions. For the second requirement, a solution could be to rasterize additional geometries at the transition level. However, such an approach increases both the rendering cost and the complexity of the rendering system. Our solution is simple and based on the following remark: the sum of weights resulting from the splatting approximately varies from 1 to 0 at the boundary of the splat representation (figures 5c-d). Therefore, the idea is to reuse these weights to perform a linear blending between the two representations. So, a high-quality hybrid rendering algorithm is obtained with only minor changes in a classical multi-pass splatting algorithm:

1. **Visibility splatting**, no change: splats are drawn as opaque ellipses into the depth buffer. The depth values of splats are shifted by ε along the view direction.
2. **Polygon rasterization**: polygons are rendered upon the previous depth buffer. Therefore, polygonal regions hidden by splats are removed and because the depth buffer is updated, splat parts hidden by polygons will also be removed during the next pass. In order to be able to reuse this result, polygons are rendered into a texture.
3. **Attribute accumulation**: splat attributes are accumulated into the attribute buffers. In order to allow the overlapping of splats and polygons of the same surface, the depth of splats and polygons must be compared with a tolerance. Then, during this pass, the depth values of splats are shifted by ε toward the viewer.
4. **Finalization**: during the normalization and deferred shading pass, the sum of weights of the splatting is clamped to $[0,1]$ and used to linearly blend the colors resulting from the splatting and polygon rasterization. Because it is difficult to ensure that the sum of weights saturates for pixels only covered by splats, this blending must be done if and only if the pixel is covered by a polygon. To do so, a solution is to check the alpha component of the polygon rendered image.

In our algorithm, we blend the resulting shaded colors. However, we notice that it is also possible to use a deferred shading strategy for the polygonal representation, and thus we can blend the attributes before the shading calculations, that should slightly increase the blending quality.

The quality of the smoothing mainly depends on the point sampling at the transition (figures 5b-d). Indeed, to get the best of our approach, it is necessary that the sum of weights saturate at the transition edges. This can be guaranteed by uniformly sampling the transition edges such that the splats share their center with the edge. It is done at the sampling step or even in a preprocess step. In the context of multiresolution rendering, a particular attention must be paid because splats located exactly on an edge must be drawn once if at least one of the polygon sharing the edge is drawn as a set of splats.

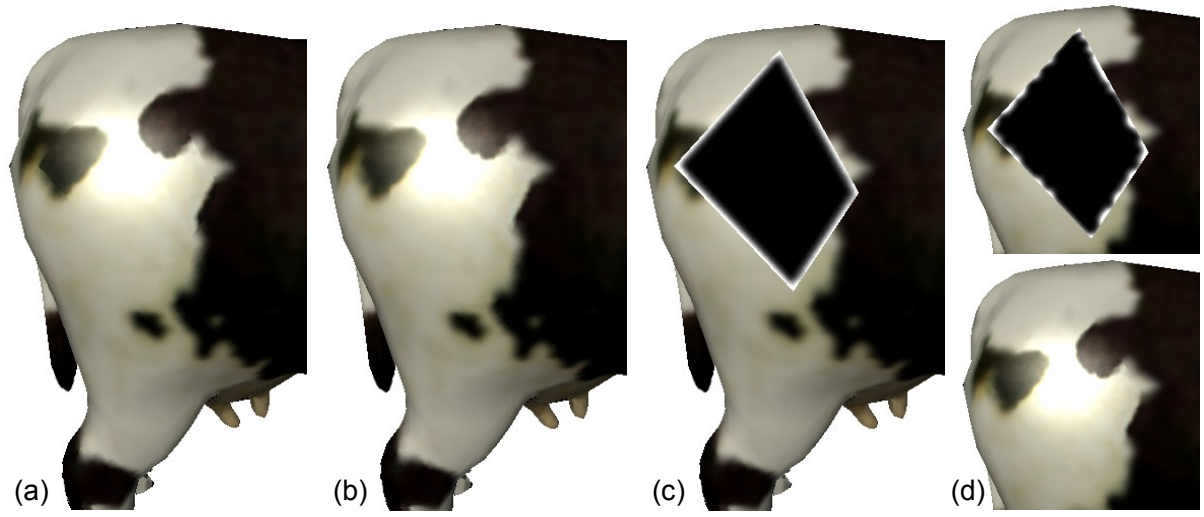


Figure 5: *Illustration of an hybrid rendering with (a) a naive approach and (b) our optimized smoothing method. Figure (c) illustrates the associated weights and (d) shows our smoothing approach without sampling optimization.*

5.2. Multiresolution rendering

In order to experiment this algorithm we have developed an hybrid multi-resolution rendering system based on an octree. Each node stores a list of splat indices while the leaves of the tree store a list of polygon indices. In a preprocessing step, each edge shared by two polygons held by two different leaves are explicitly sampled. Other edges cannot generate a transition and do not require any special treatment. In order to avoid oversampling, splats which are too close to a such edge are removed. Because the splats generated during this step are shared by two different leaves, they are indexed in a separate list where each element stores the list of splat indices shared by two different leaves. At rendering time, the octree is traversed with per node visibility culling and level-of-details estimation. If a leaf is under sampled, then the polygonal representation is selected for this part instead of the splat representation. At the end of the tree traversal, each element of the transition list is checked: if at least one of the leaves shared by the transition is drawn as a set of splats, then the list of splats held by the transition are also drawn.

6. Transparency

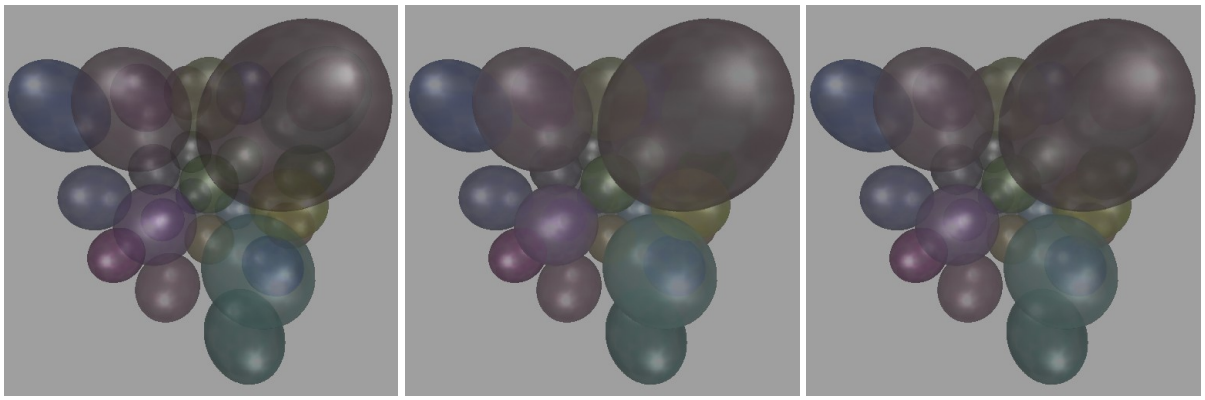
In this section we present a multi-pass approach to render transparent point-based models on current GPUs in an order independent maner. Our approach can be seen as an extention of depth-peeling techniques to deferred shaded multi-pass splatting.

Our transparency algorithm is as follow. In a first step, opaque objects are rendered by the three splatting passes: visibility, attribute accumulation and deferred shading. Next, transparent objects are drawn n times in order to get n images which respectively correspond to the different reconstructed layers sorted from the closest to the farthest of the viewer. Layers are iteratively reconstructed as follow:

1. Bind the previous depth buffer as a read-only depth texture (except for the first pass).
2. Clear the destination buffers (attributes, depth).
3. Visibility splatting. An additional depth comparison with the depth of the previous layer is performed, such that only fragments behind the previous layer pass the test.
4. Attribute accumulation. In this pass, two additional depth comparisons are added in the fragment shader: one with the depth of the previous layer and one with the depth of opaque objects such that only visible splats behind the previous layers pass these tests.
5. Normalization and deferred shading. The result of this pass is stored in a texture.

When all layers are reconstructed, they are combined with the final image by their drawing with alpha-blending in the inverse order of their construction (i.e. from the farthest to the closest).

Now, let us discuss about the choice of the number of layers n. The evaluation of the exact number of visible layers at each frame is a difficult problem. Moreover, performing a lot of passes is always inefficient and requires a lot of video memory to store the layers. Thus, we rather propose to bound the number of reconstructed layers. For instance three or four layers are sufficient for most of scenes. In this case, if the view point contains more than n visible transparent layers, the color information of unreconstructed layers will be lost, that is not satisfying (figure 6b). Hence we also propose to accumulate every visible transparent splats behind the layer $n-1$ into the last layer n (figure 6c). This is



(a) (b) (c)

Figure 6: *27 transparent point sampled spheres rendered on a common GPU with the reconstruction of (a) 6 layers at 21 fps, (b) 3 layers at 45 fps without our blending heuristic and (c) 3 layers at 49 fps but with our blending heuristic.*

realized by skipping the visibility splatting pass during the reconstruction of the last layer, that, as a bonus, slightly reduces the rendering time. In order to diminish the artifacts, we use the transparency coefficients to modulate the splatting weights, such that the most opaque surfaces will have more importance than others. Moreover, for the reconstruction of this last layer we switch back to a per-splat lighting instead of a deferred shading strategy because blending surface attributes coming from different surfaces in a such way makes no sense. This is not a real issue since the last layer is necessary far to the viewer and thus contains small splats leading to a per-pixel lighting equivalent quality. In practice, artifacts due to this approximation are seldom perceptible because the contributions of the last layers are mainly attenuated by the contributions of the first layers.

7. Results

We have experimented the different algorithms presented in this paper on an Athlon64 3500+ Linux station equipped with a GeForce 7800 graphic card.

The main contributions of our splatting implementation are both the improvement of the frame-rate and the quality of anti-aliasing in the context of perspectively correct splat rasterization. Indeed, even though our method and the one of Botsch et al. [BHZK05] often produce the same image quality (figure 3), our anti-aliasing filtering approach provides higher quality than the previous approximation for parts where the splats are both in the case of minification and magnification (figure 4). The table 1 summarizes both the vertex (VP) and fragment program (FP) complexities and performance of different splatting implementations. We notice, that the number of FP instructions reported in this table does not include the `MOV` instructions required to copy the attribute values. The VP and FP complexities can also be compared to the perspective accurate EWA splatting method [ZRB*04] where they report respectively 120 VP and 13 FP instructions for the attribute pass. Compared to our approach, the raycasting approach requires simpler vertex programs but significantly more complex fragment programs because all the splat rasterization effort is deferred to the fragment program stage. Since the number of rasterized fragments is significantly larger than the number of splats (at least a factor of four in the case of complete minification), our approach is always more performant.

Our order independent transparency rendering algorithm is illustrated figure 6 on a complex example containing a maximum of six transparent layers. As we can see, fixing the number of the reconstructed layers to three combined with our blending heuristic leads to visual results which are very close to the reference image obtained by the effective reconstruction of six layers. Finally, in the context of hybrid rep-

	Perspective splatting	raycasting [BHZK05]
#instr. visibility pass	46/3	34/9
attribute pass	58/3	35/13
154k	50 (7.7)	33 (5)
460k	40 (18.4)	26 (12)
1.4M	22 (31)	13 (18.2)
2.5M	15 (37.5)	9.2 (23)
5M	6.5 (32.5)	5 (25)

Table 1: *Our splatting algorithm is compared with the raycasting approach [BHZK05]. The top parts indicates the number of vertex and fragment program instructions required by the two splatting passes. The bottom parts reports the performances of the two implementation for the Igea data set at various sampling resolutions and for a 1024×1024 window. The first number is the number of frames per second, while the number under bracket indicates the million of rendered splats per second.*

resentations, the figure 5 shows the significant improvement in the rendering quality offered by our transition smoothing technique. Since our technique does not really require additional rendering operations, the performance remains the same than without transition smoothing.

8. Conclusion

In this paper we have presented an improved splat rendering pipeline based on a perspectively correct fast splat rasterization procedure enhanced by an approximation of the EWA filter. The efficiency of our technique comes from its very fast rasterization procedure which requires a relatively simple setup. We have also shown how to deal with transparent splat models using a GPU based implementation, without excessive overhead. Finally, in the context of hybrid rendering, we have proposed a new automatic technique blending splats and polygons at their junction, hence removing visual discontinuities. Added to existing splat rendering techniques such as sharp features rendering [ZRB*04] and real-time refinement [GBP05], the results presented in this paper yield to a more complete, flexible and high-quality splat rendering pipeline. As future works it will be interresting to improve our accurate EWA filter approximation in the case of elliptic splats.

References

[AA03] ADAMSON A., ALEXA M.: Approximating and intersecting surfaces from points. In *SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), Eurographics Association, pp. 230–239.

[AKP*05] ADAMS B., KEISER R., PAULY M., GUIBAS L. J., GROSS M., DUTRÉ P.: Efficient raytracing of deforming point-sampled surfaces. *Computer Graphics Forum 24*, 3 (2005).

[BHZK05] BOTSCH M., HORNUNG A., ZWICKER M., KOBBELT L.: High-quality surface splatting on today's GPUs. In *Proceedings of Symposium on Point-Based Graphics 2005* (2005), pp. 17–24.

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern GPUs. In *Pacific Graphics'03* (2003), pp. 335–343.

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Proceedings of Symposium on Point-Based Graphics 2004* (2004), pp. 25–32.

[CAZ01] COHEN J. D., ALIAGA D. G., ZHANG W.: Hybrid simplification: combining multi-resolution polygon and point rendering. In *IEEE Visualization 2001* (October 2001), pp. 37–44.

[CH02] COCONU L., HEGE H.-C.: Hardware-oriented point-based rendering of complex scenes. In *Proceedings Eurographics Workshop on Rendering* (2002), pp. 43–52.

[CN01] CHEN B., NGUYEN M. X.: POP: a hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001* (October 2001), pp. 45–52.

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. In *ACM Transactions on Graphics (SIGGRAPH 2003 Proceedings)* (2003), ACM Press, pp. 657–662.

[GBP04] GUENNEBAUD G., BARTHE L., PAULIN M.: Deferred splatting. In *Proceedings of Eurographics 2004* (2004), pp. 653–660.

[GBP05] GUENNEBAUD G., BARTHE L., PAULIN M.: Interpolatory refinement for real-time processing of point-based geometry. In *Proceedings of Eurographics 2005* (2005), pp. 657–666.

[GP03] GUENNEBAUD G., PAULIN M.: Efficient screen space approach for hardware accelerated surfel rendering. In *Vision, Modeling and Visualization* (Nov. 2003), IEEE Signal Processing Society, pp. 1–10.

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28*, 6 (2004), 801–814.

[KV01] KALAIAH A., VARSHNEY A.: Differential point rendering. In *Proceedings of Eurographics Workshop on Rendering Techniques 2001* (2001), pp. 139–150.

[Lev03] LEVIN D.: Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization* (2003), 181–187.

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. 141–151.

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. *ACM Transactions on Graphics (SIGGRAPH 2003 Proceedings) 22*, 3 (2003), 641–650.

[PSG04] PAJAROLA R., SAINZ M., GUIDOTTI P.: Confetti: Object-space point blending and splatting. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (September-October 2004), 598–608.

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Eurographics'02* (Sept. 2002), pp. 461–470.

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering* (June 2000), pp. 319–328.

[SP04] SAINZ M., PAJAROLA R.: Point-based rendering techniques. *Computers & Graphics 28*, 6 (2004), 869–879.

[WFP*01] WAND M., FISCHER M., PETER I., AUF DER HEIDE F. M., STRASSER W.: The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Computer Graphics* (Aug. 2001), SIGGRAPH 2001 Proceedings, pp. 361–370.

[WS05] WALD I., SEIDEL H.-P.: Interactive ray tracing of point based models. In *Proceedings of 2005 Symposium on Point Based Graphics* (2005).

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3D: an interactive system for point-based surface editing. 322–329.

[ZPvG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Computer Graphics* (2001), SIGGRAPH 2001 Proceedings, ACM Press, pp. 371–378.

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Proceedings of Graphics Interface* (2004), pp. 247–254.