View-Independent Object-Space Surface Splatting

Chun-Fa Chang* Yung-Feng Chiu† Wei-zhi Liao‡
Computer Science Department,
National Tsing Hua University, Taiwan

Figure 1: Various test models (chameleon, face, male, tiny) are rendered using view-independent object-space surface splatting.

Abstract

Elliptical weighted average (EWA) surface splatting is an important technique for producing high-quality point-based rendering. With the proliferation of programmable graphics hardware, many methods have been proposed recently to implement hardware-accelerated EWA surface splatting on modern consumer-level graphics hardware. However, we notice that those methods tend to rely heavily on customized vertex programs (or shaders) and fragment programs. Many hardware functions such as the multisampling and anisotropic texture filtering features that could be equally useful for both triangle-based rendering and point-based rendering are seldom put to use in current EWA surface splatting methods. In this work, we propose a simple yet effective change to the EWA surface splatting method by removing the low-pass filter from the resampling process. The low-pass filtering is then taken care of by multisampling (for full-scene antialiasing). This simple change effectively extracts the screen-space issues out of the EWA surface splatting problem. Therefore, the remaining parts of EWA surface splatting become purely object-space problems which are much easier to solve. For example, perspective accurate splatting which requires special attention in some screen-space surface splatting algorithms is easily achieved as long as the hardware supports perspective-correct interpolation. Other hardware features such as anisotropic texture filtering can be easily added as well. Our results show the rendering quality is comparable to those produced with resampling filters with tightly-coupled low-pass filters. We also achieve rendering speed of about 6 million splats per second.

*e-mail: chunfa@cs.nthu.edu.tw
†e-mail: yfchiu@ibr.cs.nthu.edu.tw
‡e-mail: demon@ibr.cs.nthu.edu.tw

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms—; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics Data Structures and Data Types—;

Keywords: graphics hardware, antialiasing, point-based rendering

1 Introduction

With the recent advances in image-based modeling and 3D scanning techniques, point-based models have become an alternative or even the preferred choice to triangle-based models in many applications. Not too long (about 3-5 years) ago, the traditional triangle-based rendering pipeline was not considered flexible enough to produce hardware-accelerated high-quality point-based rendering. Special hardware features for point-based rendering such as those in [Popescu et al. 2000] were then proposed. With the proliferation of programmable graphics hardware, high-quality point-based rendering based on elliptical weighted average (EWA) surface splatting [Zwicker et al. 2001] [Ren et al. 2002] is now possible on modern graphics hardware. However, we notice that many hardware functions (such as the antialiasing and texture filtering features) that are originally designed for triangle-based rendering are under-utilized for point-based rendering. For examples, the multisampling (for antialiasing) and anisotropic texture filtering [McCormack et al. 1999] [Schilling et al. 1996] features could be equally useful for both triangle-based rendering and point-based rendering, but they are seldom put to use in current EWA surface splatting methods.

In this work, we propose a simple change to the EWA surface splatting method by removing the low-pass filter from the resampling process. The low-pass filtering is then taken care of by the multisampling (or full-scene antialiasing) feature. Other hardware features such as anisotropic texture filtering can be easily added as well for removing aliasing artifact of texture mapping objects. While the idea is simple, it effectively extracts the screen-space issues out of the EWA surface splatting problem. Therefore, the remaining parts of EWA surface splatting become a purely object-space problem which is much easier to solve. For example, it is much easier to produce a perspective-accurate splatting in object space without having to choose between either accurate outline or accurate center,

a problem well described in Figure 2 of [Zwicker et al. 2004].

Our results show the rendering quality is comparable to those produced by the original EWA surface splatting methods (with low-pass filters tightly-coupled to resampling filters). We also achieve rendering speed of about 6 million splats per second.

2 Related Work

Surfels [Pfister et al. 2000] is an important data representation for point-based graphics. Surfels are usually stored in Layered Depth Cubes (LDC) which can then be extended to a hierarchical representation [Chang et al. 1999] [Pfister et al. 2000] [Rusinkiewicz and Levoy 2000] [Botsch et al. 2002] [Dachsbacher et al. 2003]. An LDC consists of three orthogonally projected Layered Depth Images (LDI). A limitation of the LDI is that it does not work well with surfaces at grazing angles. If those surfaces are sampled from the LDI grid points, they will be undersampled. If they are sampled from another view and then warped to the LDI, some LDI pixels will inevitably contain large number of layers. In an LDC, we do not have those problems because the surfaces are sampled from the grid points of three LDIs. A surface might be undersampled in one LDI (or even two LDIs) but not in all three LDIs. However, by sampling a surface three times, an LDC may consume nearly three times as much memory as an LDI.

Surfel-based models are usually rendered by surface splatting [Zwicker et al. 2001]. Based on the theorem of signal processing, surface splatting produces output images of extremely high quality. Surface splatting reconstructs the surfaces by conceptually considering surfels as tangent-space ellipses with Gaussian alpha fall-offs. During rendering, aliasing may occur when some of those ellipses are projected to small areas in screen space. For example, some ellipses might be so small that they fall between screen pixel grids and are totally missed during resampling. Therefore, a low-pass pre-filter must be applied when those ellipses are rendered and resampled in screen space. In addition, a separate rendering pass which is often called visibility splatting is necessary to prevent the occluded surfels from being incorrectly blended.

Ren et al. introduced an object-space surface splatting method in [Ren et al. 2002]. In their method, a texture-mapped polygon that represents the resampling filter is constructed for each surfel in object space. The texture-mapped polygon is formed by warping the screen-space low-pass filter to the object space and then convolving with the reconstruction filter (i.e. the tangent-space ellipse). Due to the low-pass filter, the construction of surfel polygons becomes a view-dependent problem in the object-space surface splatting method. This prevents the surfel polygons to be constructed at a preprocessing step.

In Ren's method and this work, four vertices per splat are presented to the graphics hardware. Rendering methods using one vertex per splat was presented by Botsch and Kobbelt [2003] and Guennebaud and Paulin [2003]. Those methods use affine approximation to the projective mapping to compute the splatting footprint in screen space. This simplification may lead to holes in the image, when the model is viewed from extremely flat angles and is shifted away from the main view axis. This problem was addressed by Zwicker et al. [2004], but they still need to choose between either accurate outline or accurate center. Most of those methods need to correct the depth value per pixel which bounds their rendering performance.

Hierarchical representations for point clouds [Chang et al. 1999] [Pfister et al. 2000] [Rusinkiewicz and Levoy 2000] [Botsch et al. 2002] [Coconu and Hege 2002]can improve the rendering efficiency. However, those implementations are all software-based and add a heavily load on CPU. Dachsbacher et al. [2003] present sequential point trees (SPT), a hierarchical point representation allowing adaptive rendering of point clouds completely on the graphics processor unit (GPU) sequentially. But their approach draws each splat as a square, hence the quality could be improved by applying Gaussian-like filters.

The above mentioned methods vary in performance and quality of rendering. The reported performance ranges from 2-3M splats/sec [Ren et al. 2002] to about 50M splats/sec [Dachsbacher et al. 2003]. However, many of them sacrificed rendering quality for achieving better performance. In Table 1, we compare their rendering quality based on whether correct splat shape, alpha blending, low-pass filter, visibility splatting are used. Our method achieve rendering quality that is similar to [Ren et al. 2002] at a faster speed.

3 Object-Space EWA Surface Splatting

In this section, we briefly review the object-space EWA surface splatting. We use the same notations as in [Ren et al. 2002]. First, we focus on screen-space EWA surface splatting. Let $\{P_k\}$ be the set of points in 3D object space without connectivity. It is associated with a radially symmetric basis function r_k as a reconstruction filter defined on a locally parameterized domains and a coefficient $w_k(w_k^r, w_k^g, w_k^b)$ that represent continuous functions for red, green and blue color components. We define the continuous surface function $f_c(u)$ as the weighted sum:

$$f_c(u) = \sum_{k \in N} w_k r_k(u - u_k) \tag{1}$$

Given an input function as in Equation (1) and an affine approximation of the perspective projection from parameter space to screen space at each point $x = m(u) : R^2 \rightarrow R^2$, $f_c(u)$ is warped to screen space by m, yielding the continuous screen space output function $g_c(x)$:

$$g_c(x) = f_c \circ m^{-1}(x) = f_c(m^{-1}(x)) \tag{2}$$

Then the continuous screen-space output function $g_c(x)$ is band-limited by convolving it with a prefilter h, yielding the output function $g'_c(x)$:

$$g'_c(x) = g_c(x) \otimes h(x) \tag{3}$$

Finally, we sample the continuous output function $g'_c(x)$ by multiplying it with an impulse train i to produce the discrete output $g(x)$:

$$g(x) = g'_c(x) i(x) \tag{4}$$

We replace a projection mapping $m(u)$ by its local affine approximation m_k at a point u_k and write the output function $g'_c(x)$ as a weighted sum of screen space resampling filters $\rho_k(x)$:

$$g'_c(x) = \sum_{k \in N} w_k \rho_k(x) \tag{5}$$

where

$$\rho_k(x) = (r'_k \otimes h)(x - m_k(u_k)) \tag{6}$$

Hence, the resampling filter $\rho_k(x)$ is expressed as a convolution of a warped basis function $r'_k = r_k(m^{-1}(x))$ and the prefilter $h(x)$. The local affine approximation m_k is given by the Taylor expansion of m at u_k, truncated at the linear term:

$$m_k(u) = x_k + J_k \cdot (u - u_k) \tag{7}$$

Method	splats/sec	splat shape	alpha blending	low-pass filter	visibility splatting
[Ren et al. 2002]	2-3M	ellipse	Yes	Yes	Yes
Our method	5-6M	ellipse	Yes	Yes	Yes
[Botsch and Kobbelt 2003]	10M	ellipse	Yes	No	Yes
[Dachsbacher et al. 2003]	50M	square	No	No	No

Table 1: Rendering quality and performance comparison.

where $x_k = m(u_k)$ and the Jacobian $J_k = \frac{\partial m}{\partial u}(u_k) \in R^{2x2}$. In Equation (6), we have expressed the screen space resampling filter. Now, we rearrange it to get the object-space resampling filter $\rho'_k(x)$:

$$\begin{aligned} \rho_k(x) &= (r'_k \otimes h)(x - m_k(u_k)) \\ &= (r'_k \otimes h)(m_k(m_k^{-1}(x)) - m_k(u_k)) \\ &= (r'_k \otimes h)(J_k(m_k^{-1}(x) - u_k)) \\ &= (r_k \otimes h')(u - u_k)) = \rho'_k(x) \end{aligned} \tag{8}$$

The object-space resampling filter consists of the convolution of an original radially symmetric basis function $r_k(u)$ and a warped prefilter $h'_k(u) = |J_k| h(J_k(u))$. As in [Ren et al. 2002], elliptical Gaussians are chosen for both the basis functions and the low-pass filter. Gaussians are closed under affine mapping and convolution, hence the resampling filter can be expressed as a single Gaussian. Let $G_v(x)$ be a Gaussian with variance matrix V:

$$G_v(x) = \frac{1}{2\pi |V|^{\frac{1}{2}}} e^{-\frac{1}{2} x^T V^{-1} x} \tag{9}$$

where $|V|$ is the determinant of V. A typical choice for variance matrix of prefilter h is the identity matrix I. Let V_k^r be the variance matrix of the basis functions r_k, then the object-space EWA resampling filter is shown as:

$$\rho'_k(u) = G_{V_k^r + J_k^{-1} J_k^{-1^T}}(u - u_k) \tag{10}$$

Finally, we use Equation (10) to reformulate the continuous function as shown in Equation (5) as a weighted sum of object-space EWA resampling filter:

$$g'_c(x) = \sum_{k \in N} w_k G_{V_k^r + J_k^{-1} J_k^{-1^T}}(m^{-1}(x) - u_k) \tag{11}$$

4 View-Independent Object-Space Surface Splatting

Given the widespread use of surfels and surface splatting, we take a closer look at various design choices in the surfel representation and its rendering. We think it is possible to make surfel quads view independent if we can eliminate the need for the low-pass filter. Our approach is simple: we omit the low-pass filter and rely on the full-scene antialiasing (multisampling) feature of modern graphics hardware. In a sense, the low-pass filter is not eliminated. Its role is simply assumed by the multisampling function of graphics hardware. Our object space resampling filter is shown as:

$$\rho_k(u) = G_{V_k^r}(u - u_k) \tag{12}$$

Then, we substitute Equation (12) to the continuous function as shown in Equation (2) as a weighted sum of object-space resampling filter:

$$g_c(x) = \sum_{k \in N} w_k G_{V_k^r}(m^{-1}(x) - u_k) \tag{13}$$

where m is a perspective projection mapping.

Finally, we take sub-samples at the pixel level to solve the aliasing problem.

$$g'(x) = \sum_{i \in N} w_i g(x_i) \tag{14}$$

where $g(x_i)$ is the i^{th} sub-pixel with the weight w_i.

The above is performed by graphics hardware that supports multisampling. The number of subsamples and how the subsamples are filtered (with w_i) to produce the pixel color vary with each hardware. Since one purpose of the multisampling function is to approximate the low-pass filter for proper full-scene antialiasing, it can practically replace the the prefilter $h(x)$ in Equation (3).

For texture-mapped point-based objects, we can also rely on the anisotropic texture-filtering feature that becomes common in graphic hardware. Rendering texture-mapped point-based models is different from rendering surfels that contain pre-filtered texture colors [Zwicker et al. 2001]. For example, if a splat covers many screen pixels (or fragments), each rendered pixel (fragment) will trigger a texture evaluation that may require a texture filtering from a footprint area.

Again, having view-independent surfel polygons in object space greatly simplifies our problem. We simply apply two textures to the surfel polygons: one texture for the reconstruction kernel (with alpha fall-off) and the other texture for the image texture. When the surfel polygon is rasterized, the anisotropic texture filtering function of graphics hardware automatically computes the appropriate footprint area in the texture space for each rendered fragment. Figure 2 compares the rendering quality produced by our method using hardware multisampling and anisotropic texture filtering to the quality produced by EWA surface splatting. In the following sections, we describe the detail of our point-based rendering pipeline.

4.1 Construction of Surfels

First, we generate surfels and store them in an LDC by ray tracing. Rays are created for LDI grid points by orthogonal projection. Each ray-surface intersection creates a surfel, so we record its depth, color (or texture coordinates), and normal vector. We do not build a LDC tree because our purpose is to investigate the quality of our proposed rendering algorithms. Then, we assign surfels to LDC faces based on the maximum magnitude of the N_x, N_y, N_z coordinates of the normal vectors which is different from the 3-to-1 reduction described in [Pfister et al. 2000] and the redundant surfel elimination technique described in [Zwicker 2003]. An 2D example is shown in Figure 3. As we can see in Figure 3(b), the resampling process alters the positions of the point samples. Also, for surfaces that form grazing angles to the LDI plane, the sample points concentrate on only a few LDI grid points, thus produce long lists of pixels. In our representation, the normal vectors tell us which of the three LDIs has a better view of the surface. We still use three LDIs in each LDC, but each LDI now contains roughly one third of the sample points. Figure 3(c) shows an 2D example of our proposed

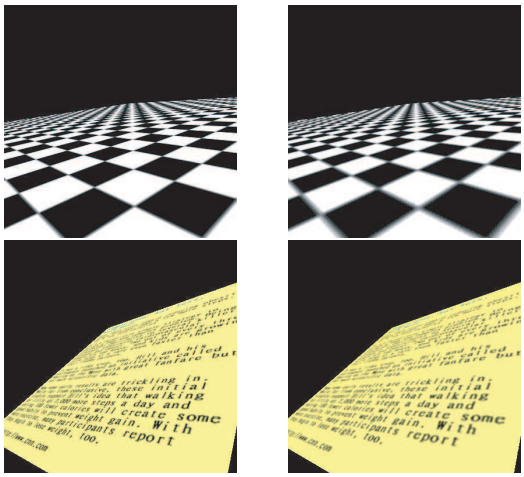

Figure 2: **Left:** The checker and text boards rendered using multisampling (sample=4) and anisotropic texture filtering (anisotropy=8). **Right:** The same models rendered using EWA surface splatting.

representation. Table 2 shows the number of surfels for each surfel elimination technique. From the comparison of each reduction technique in Figure 4, our surfel representation makes LDC more memory efficient while maintaining the same image quality. Figure 5 shows another comparison of rendering quality among these reduction techniques.

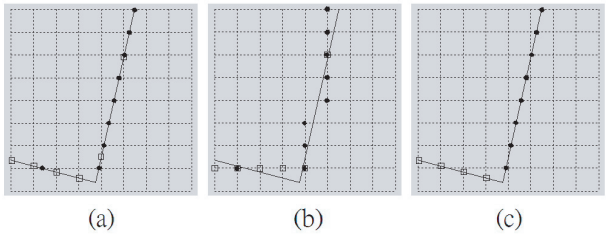

Figure 3: These figures show in 2D how surface sample points are stored in an LDC. Black dots are the samples that are stored in the right-side LDI and white squares are stored in the bottom-side LDI. (a) The original surfel representation without any reduction. (b) The original surfel representation after 3-to-1 reduction. (c) Our proposed representation.

Model	Original	3-to-1	Zwicker's	Our
Tiny	311,028	180,947	271,682	177,782
Tiger	206,064	140,203	180,282	136,143
Butterfly	251,897	212,943	251,480	224,144

Table 2: Number of surfels in four different LDC representations.

Finally, we extend the surfel to a surfel quad and its size and orientation are determined from the LDI grid resolution (which represents the sampling density) and its normal vector for each LDI separatively. Figure 6 illustrates an example to construct surfel quad for the surfel belonging to XY-plane of the LDC with position $P(X, Y, Z)$, normal $N(N_x, N_y, N_z)$ under the sampling density H.

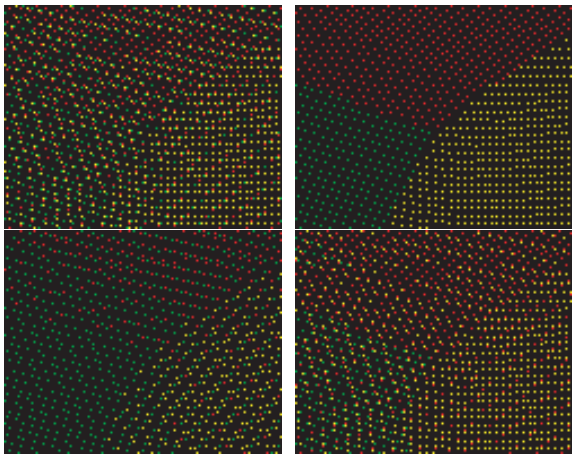

Figure 4: LDC representation comparison for a realistic model among three reduction techniques (ours, 3-to-1, Zwickers). Red, green and blue colors show 3 directional LDI separately. **Top-Left:** Without any reduction. **Top-Right:** Our normal reduction. **Bottom-Left:** 3-to-1 reduction. **Bottom-Right:** Zwicker's normal reduction causing an overlap region.

$$
\begin{aligned}
P_1 &= (X+H/2, Y-H/2, (Z+(N_x/N_z - N_y/N_z)\times H)) \\
P_2 &= (X-H/2, Y-H/2, (Z+(-N_x/N_z - N_y/N_z)\times H)) \\
P_3 &= (X-H/2, Y+H/2, (Z+(-N_x/N_z + N_y/N_z)\times H)) \\
P_4 &= (X+H/2, Y+H/2, (Z+(N_x/N_z + N_y/N_z)\times H))
\end{aligned}
$$

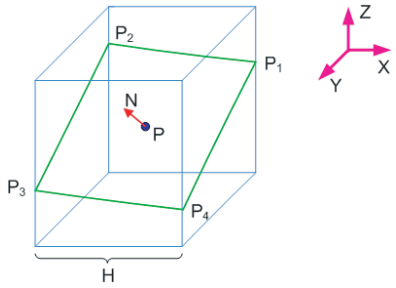

Figure 6: Construct a surfel quad in object space.

4.2 Visibility Splatting

To correctly accumulate the filter weights from surfels and reconstruct the represented surfaces, additive alpha blending is used. However, without hidden surface removal, the filter weights from occluded surfaces would be incorrectly accumulated too. Therefore, surface splatting requires an extra pass called visibility splatting before the normal rendering pass. Our visibility splatting method is similar to [Ren et al. 2002]. During this pass, it fills the depth buffer by rendering the surfel quad (with a circular mask map) slightly shifted away from the viewer by an offset ε.

4.3 Color Splatting

To render a surfel, we convert a surfel into an object-space quad with a 2D Gaussian alpha map that represents the reconstruction filter. If the surface is not textured, then the surfel color is modulated by the alpha texture map. Our approach is similar to the

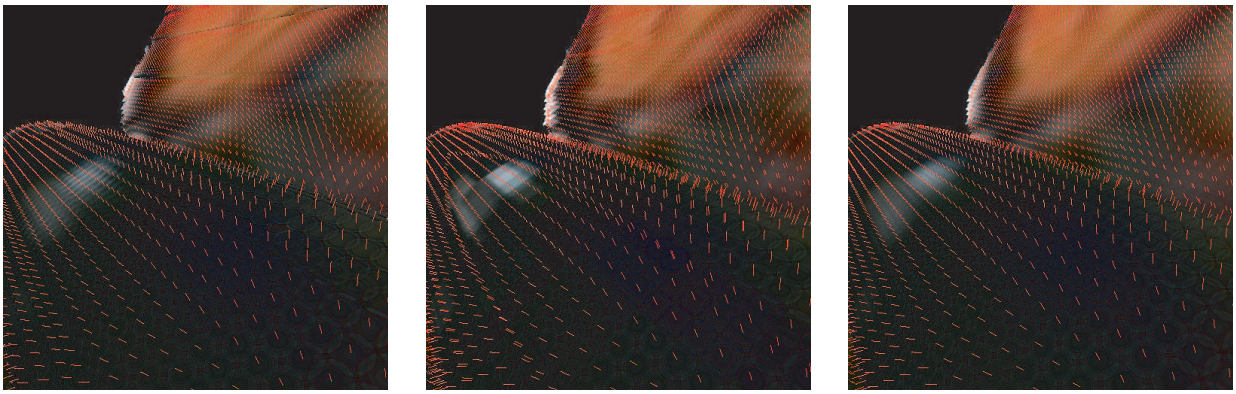

Figure 5: Comparison of each reduction techniques for Butterfly model. **Left:** 3-to-1 reduction. **Middle:** Zwicker's normal reduction. **Right:** Our normal reduction.

object-space EWA surface splatting [Ren et al. 2002] except that our surfel quads can be built at a pre-processing step. In [Ren et al. 2002], they warp the screen-space low-pass filter to the object space and convolve it with the reconstruction filter. We omit that step because we use the full-scene antialiasing (multisampling) feature of graphics hardware to do the low-pass filtering. That makes the construction of surfel quads a view-independent problem, which could be performed at pre-processing time.

As pointed out by [Zwicker et al. 2004], Gaussians are closed under affine mapping, but not under projective mapping. Since we omit the step to warp the screen-space low-pass filter to the object space, our surfel quad contains perspective-accurate projection of a Gaussian map, which is demonstrated in Figure 7. It is easily achieved in our method as long as the graphics hardware supports perspective-correct interpolation, yet another benefit of our view-independent framework.

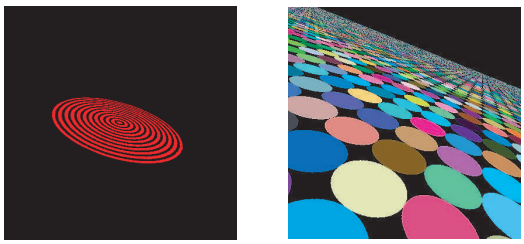

Figure 7: **Left:** Perspective correct center. **Right:** Our approach leads to perspective correct splat shape.

4.4 Texture Mapped Point-Based Objects

For textured surfaces, we apply two textures (one is the alpha texture map and the other is the model's texture image) to the surfel quads. There is difference between rendering textured-mapped surfels and rendering surfels that pre-filter the texture and store the filtered colors with the surfels. When a surfel covers more than one fragment on the screen, each fragment will trigger a texture lookup during rasterization. Therefore, it is natural to rely on the graphics hardware to obtain a well-filtered texture color for the fragment. Nowadays, anisotropic texture filtering is common in graphics hardware. It dramatically improves visual quality where texture minification in the s and t directions is radically different.

4.5 Weight Normalization

Ideally, with hidden surface removal and additive alpha blending, the sum of filter weights equal to one for pixels in screen space. Unfortunately, the reconstruction kernels in screen space do not always form a partition of unity, which causes visible artifacts. As discussed in [Ren et al. 2002], the accumulated filter weights do not always sum up to one in general. There are two main reasons that the reconstruction kernels do not form a partition of unity. First, we usually use a Gaussian filter that is truncated to a finite support. Second, the reconstruction kernel operates on the tangent space. That does not bode well for surface areas with high curvature. We only perform per-pixel normalization on those pixels whose summation of filter weights is not close to unity or larger than a threshold ε to avoid edge aliasing for silhouette edge pixels.

We know that the RGB color model is based on a Cartesian coordinate system, the color subspace of interest is the unit cube. However, the normalization operation will change the pixel color due to nonlinear operation "div". In spite of the fact that the accumulated filter weights do not always sum up to one in general, our experiments show that our accurate splat shape and center accumulate the filter weights more closely to a partition of unity than the previous affine approximations. More on this when we present our results (Figure **??**).

5 Implementation

We have implemented both our method and the algorithms in [Ren et al. 2002] with OpenGL. All vertex and fragment programs are implemented using ARB_vertex_program and ARB_fragment_program hardware vender independent extensions or using the more platform-independent language Cg [Mark et al. 2003]. We also put the geometries of the model into video memory via ARB_vertex_buffer_object extension to avoid traffic load between CPU and GPU for each drawn primitive. In order to utilize the hardware vertex cache function we build an extra index buffer and arrange the geometry data in interleaved vertex layout. We construct surfel quads during pre-processing using the formulas describing in Figure 6, with each vertex containing attributes such as normals and colors (or texture coordinates) in addition to its position.

Our rendering algorithm has three passes: visibility splatting pass, color splatting pass and normalization pass. In visibility splatting pass, the vertex program compute the depth offset in view space and

we also turn on alpha test function to discard pixels outside the elliptical. The Cg shader program for visibility splatting computation is listed as follows:

$$p.xyzw = mul(model\_view\_matrix, IN.position);$$
$$p.xyz = normalize(p.xyz) * (length(p.xyz) + \varepsilon);$$
$$OUT.position = mul(projection\_matrix, p);$$

In color splatting pass, we turn on multisampling function via ARB_multisample extension and accumulate the weighted contributions of splat using blending modes in the form of $(\sum_i \alpha_i(rgb)_i, \sum_i \alpha_i)$ via EXT_blend_func_separate extension. For texture-mapped surfaces, we also turn on anisotropic texture-filtering to automatically compute the appropriate footprint area in the texture space for each rendered fragment via EXT_anisotropic_filter extension. Except the reasons described in Section 4.5, the rendering quality also relies on the floating-point precision and saturation-free pixel accumulation but the alpha blending function on current generation of GPUs is limited to fixed-point precision. Hence, in the normalization pass, we perform per-pixel normalization on those pixels whose summation of filter weights is not close to unity and is greater than a threshold ε to avoid artifacts in silhouette edge pixels.

In order to allow non-power-two window-sized rectangle, we use EXT_texture_rectangle extension. Table 3 shows the instruction numbers of Cg shader program at each pass. Lower instruction numbers imply potentially more rendering speed.

6 Results

The results we present are all measured on a PC running Windows XP with a 2.4GHz Pentium 4 processor, 512 MB RAM, AGP 8X and a ATI Radeon 9600 Pro GPU. In order to test the antialiasing quality in our method, we render a checkerboard and a text-board to compare with those produced with resampling filters containing tightly-coupled low-pass filters (Figure 10).

As shown previously in Figure 7, our surfel quads contain Gaussian alpha maps that are correctly projected to screen space, leading to both the correct outer splat contour and the correct splat center for any viewing position. The reconstructed continuous screen-space output function as expressed in Equation (13) is more accurate than those by using affine approximation approach. As illustrated in [Zwicker et al. 2004], incorrect splat shape may lead to holes in the rendered image when the model is viewed from a flat angle and shifted sufficiently from the viewing axis. This artifact is shown in Figure 9 for those extreme viewing positions. Also, the incorrect splat center and the overlapped splat shape caused by the affine approximation affect the color distribution within the splat. This is the reason why the resampling filters using affine approximation favor over-blurring the colors over aliasing artifacts, as shown in Figure 8.

Finally, we compare the rendering performance between our method and Ren's method in Table 4. We render 4M to 6M surfels per second which is twice of Ren's approach. Figure 1 shows more rendering results of some complex textured models.

7 Conclusions and Future Work

Object-space EWA surface splatting [Ren et al. 2002] allows high-quality surface splatting [Zwicker et al. 2001] to take advantage of the graphics pipeline. In this work, our approach fully exploits the capabilities of graphics hardware for point-based rendering by utilizing the multisampling and anisotropic texture filtering features. This leads to a larger common hardware feature set for both point-based rendering and triangle-based rendering, which could simply the hardware design in the future. Our method also leads to simpler and shorter vertex programs and fragment programs, which translates to better performance.

We could further embed hierarchical representations into our framework to improve the splat rate. Our method can also be extended to support animated point-based models. For next generation of graphic hardware, floating-point precision may become available for alpha blending. That will allow us to further improve the quality of rendered images.

Figure 8: Real-world models (male and chameleon) illustrate Ren's approach favoring over-blurring the texture over aliasing artifacts. **Left:** Models are rendered with multisampling (sample=4). **Right:** Models are rendered using Ren's approach.

References

BOTSCH, M., AND KOBBELT, L. 2003. High-quality point-based rendering on modern gpus. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 335–346.

BOTSCH, M., WIRATANAYA, A., AND KOBBELT, L. 2002. Efficient high quality rendering of point sampled geometry. In *Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, 53–64.

CHANG, C.-F., BISHOP, G., AND LASTRA, A. 1999. Ldi tree: a hierarchical representation for image-based rendering. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 291–298.

COCONU, L., AND HEGE, H.-C. 2002. Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, 43–52.

Method	Visibility splatting		Color splatting		Normalization	
	VP	FP	VP	FP	VP	FP
Our method	15	-	5	-	-	8
[Ren et al. 2002]	17	-	74	-	-	8
[Zwicker et al. 2004]	151	28	164	22	-	4
[Guennebaud and Paulin 2003]	29	5	45	5	-	3

Table 3: Comparison of the number of instructions needed for each pass.

Model	Points	Texture-mapped	Our		[Ren et al. 2002]	
			(512^2)	(1024^2)	(512^2)	(1024^2)
Face	40,880	No	112	66	36	32
Chameleon	101,685	No	53	41	16	14
Tiger	136,143	Yes	40	32	12	11
Male	148,138	No	37	30	11	11
Tiny	177,782	Yes	32	27	9	8
Butterfly	224,144	Yes	26	22	8	8
Checkerboard	263,682	Yes	22	20	6	5

Table 4: Performance comparison between our and Ren's approaches (frame per second).

Figure 9: **Left:** Our approach is hole-free. **Right:** Affine approximations lead to holes in the reconstruction for extreme view positions.

DACHSBACHER, C., VOGELGSANG, C., AND STAMMINGER, M. 2003. Sequential point trees. *ACM Trans. Graph. 22*, 3, 657–662.

GUENNEBAUD, G., AND PAULIN, M. 2003. Efficient screen space approach for Hardware Accelerated Surfel Rendering. In *Vision, Modeling and Visualization, Munich*, IEEE Signal Processing Society, 1–10.

MARK, W. R., GLANVILLE, R. S., AKELEY, K., AND KILGARD, M. J. 2003. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. Graph. 22*, 3, 896–907.

MCCORMACK, J., PERRY, R., FARKAS, K. I., AND JOUPPI, N. P. 1999. Feline: fast elliptical lines for anisotropic texture mapping. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 243–250.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 335–342.

POPESCU, V., EYLES, J., LASTRA, A., STEINHURST, J., ENGLAND, N., AND NYLAND, L. 2000. The warpengine: an architecture for the post-polygonal age. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 433–442.

REN, L., PFISTER, H., AND ZWICKER, M. 2002. Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. *cgforum 21*, 3 (Sept.). (Proc. Eurographics 2002).

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 343–352.

SCHILLING, A., KNITTEL, G., AND STRASSER, W. 1996. Texram: A smart memory for texturing. *IEEE Comput. Graph. Appl. 16*, 3, 32–41.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 371–378.

ZWICKER, M., RÄSÄNEN, J., BOTSCH, M., DACHSBACHER, C., AND PAULY, M. 2004. Perspective accurate splatting. In *Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, 247–254.

ZWICKER, M. 2003. *Continuous reconstruction, rendering, and editing of point-sampled surfaces*. Ph.D. Dissertation. No. 15135, University of ETH Zurich, Department of Computer Science.

Model	Points	(512^2)				(1024^2)			
		Normal	-Pass1	-Pass2	-Pass3	Normal	-Pass1	-Pass2	-Pass3
Face	40,880	112	188	178	130	66	87	91	96
Chameleon	101,685	53	96	90	56	41	63	61	52
Tiger	136,143	40	76	70	43	32	50	52	39
Male	148,138	37	70	66	39	30	47	49	36
Tiny	177,782	32	60	56	33	27	42	43	31
Butterfly	224,144	26	54	45	27	22	38	37	24
Checkerboard	263,682	22	45	38	23	20	36	32	22

Table 5: Performance analysis for our approach (frame per second).

Model	Points	(512^2)				(1024^2)			
		Normal	-Pass1	-Pass2	-Pass3	Normal	-Pass1	-Pass2	-Pass3
Face	40,880	36	44	166	38	32	37	99	36
Chameleon	101,685	16	18	79	17	14	16	60	15
Tiger	136,143	12	14	60	12	11	13	49	12
Male	148,138	11	13	56	12	11	12	45	11
Tiny	177,782	9	11	47	10	8	10	40	9
Butterfly	224,144	8	9	38	8	8	9	33	8
Checkerboard	263,682	6	8	33	6	5	6	29	5

Table 6: Performance analysis for Ren's approach (frame per second).

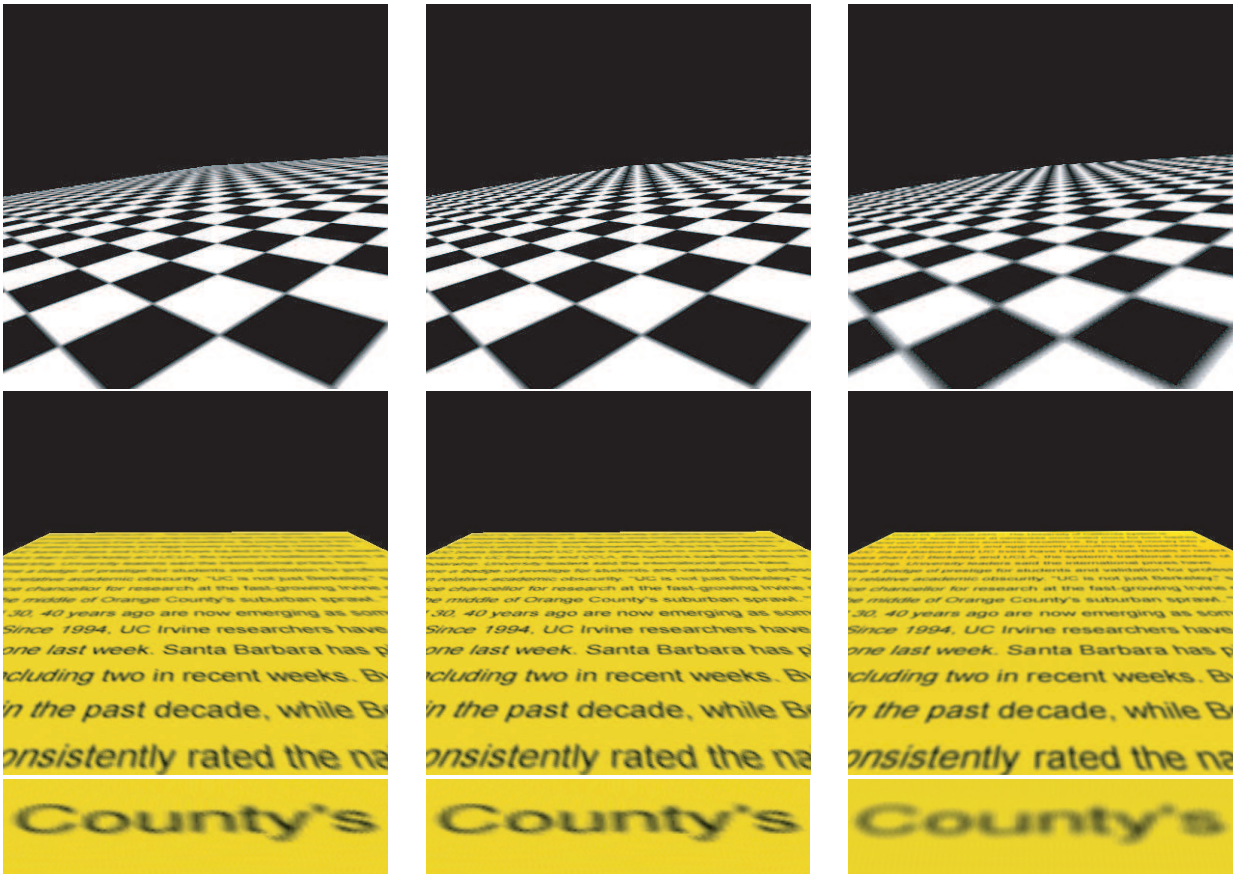

Figure 10: The image quality comparison between our and Ren's approach. **Left:** The model rendered without multisampling and anisotropic texture filtering. **Middle:** The model rendered with multisampling (sample=4) and anisotropic texture filtering (anisotropy=8). **Right:** The model rendered using Ren's approach.