Eurographics Symposium on Point-Based Graphics (2005)
M. Pauly, M. Zwicker, (Editors)

High-Quality Surface Splatting on Today's GPUs

Mario Botsch[1], Alexander Hornung[1], Matthias Zwicker[2], Leif Kobbelt[1]

[1] Computer Graphics Group, RWTH Aachen Technical University
[2] Computer Graphics Group, Massachusetts Institute of Technology

Abstract

Because of their conceptual simplicity and superior flexibility, point-based geometries evolved into a valuable alternative to surface representations based on polygonal meshes. Elliptical surface splats were shown to allow for high-quality anti-aliased rendering by sophisticated EWA filtering. Since the publication of the original software-based EWA splatting, several authors tried to map this technique to the GPU in order to exploit hardware acceleration. Due to the lacking support for splat primitives, these methods always have to find a trade-off between rendering quality and rendering performance.
In this paper, we discuss the capabilities of today's GPUs for hardware-accelerated surface splatting. We present an approach that achieves a quality comparable to the original EWA splatting at a rate of more than 20M elliptical splats per second. In contrast to previous GPU renderers, our method provides per-pixel Phong shading even for dynamically changing geometries and high-quality anti-aliasing by employing a screen-space pre-filter in addition to the object-space reconstruction filter. The use of deferred shading techniques effectively avoids unnecessary shader computations and additionally provides a clear separation between the rasterization and the shading of elliptical splats, which considerably simplifies the development of custom shaders. We demonstrate quality, efficiency, and flexibility of our approach by showing several shaders on a range of models.

1. Introduction

In the last five years, point-based surface representations have proven to be a flexible and efficient alternative to mesh-based surface representations. Directly working on point-sampled geometries greatly simplifies the 3D content creation and surface reconstruction process, as no connectivity information has to be generated and no topological manifold constraints have to be taken care of. Since there is also no need to maintain consistency during surface modifications, algorithms which require frequent re-structuring or re-sampling of surfaces benefit the most from point-sampled surfaces.

When considering point-based surface representations in general, we further distinguish between a piecewise constant point sampling [PZvBG00, ABCO*01] and piecewise linear surface splats [ZPvBG01]. In this paper we focus on surface splats, since besides providing a higher approximation order, they also allow for more efficient rendering and achieve a higher visual quality by sophisticated anti-aliasing techniques [KB04].

One key component for any interactive application processing point-based surfaces is a suitable point or splat rendering method. On the one hand, performance is a major goal, since otherwise the rendering might limit the flow of the user interaction. On the other hand, many applications require a high visual quality, e.g., in order to rate the quality of a surface based on its specular shading or reflection lines. Since both goals are typically conflicting, a trade-off has to be found in an application-dependent manner.

At one end of this quality-vs-performance scale, the original surface splatting [ZPvBG01] is located. This approach employs per-pixel lighting and thus achieves results comparable to Phong shading if the surface is sampled sufficiently densely. Aliasing artifacts are effectively avoided by EWA filtering, which is conceptually similar to anisotropic texture filtering. However, the original renderer is purely software-based and therefore limited to about 1M splats/sec on a 3.0GHz Pentium4 CPU, which is by far not sufficient for the massive datasets to be processed in many current applications.

© The Eurographics Association 2005.

To overcome these limitations, several authors tried to implement EWA splatting on graphics hardware, but — depending on the generation of GPUs available at that time — certain compromises had to be made, in terms of either rendering performance or visual quality. While each new graphics hardware typically increases the rendering performance and thus improves the rendering results quantitatively, the jump to the latest GPU generation additionally yields a significant qualitative improvement.

In this paper we show how to exploit the increased capabilities of latest graphics hardware for GPU-based surface splatting, such that the trade-off between quality and efficiency is effectively minimized. The availability of multiple render targets with floating point precision and blending capabilities now finally enables us to implement all computations required for high-quality surface splatting directly on the GPU. Hence, this paper does not introduce genuinely new concepts, but focuses on the efficient implementation of a hardware-accelerated deferred shading framework and a simple and effective approximation of the EWA pre-filter to achieve fast and high-quality surface splatting:

- **Deferred shading** allows us to closely follow the original EWA splatting approach and perform high-quality per-pixel Phong shading. Lighting is computed only once for each pixel of the final image, instead of once for each generated fragment. This is especially important for splat-based rendering, since due to the required overlap of individual splats, the number of generated fragments is much larger than the number of resulting image pixels. The use of floating point render targets furthermore avoids commonly observed shading artifacts due to discretization problems. We will prove the quality and efficiency of our deferred shading approach on complex shadow-mapped Phong and NPR shaders.
- **EWA approximation.** The original EWA filtering method is based on a composition of an object-space reconstruction filter with a screen-space pre-filter. Many previous approaches omit the screen-space filter and sacrifice anti-aliasing quality for higher rendering performance. We propose a simple approximation to the EWA pre-filter, which can be computed efficiently, but still provides high-quality anti-aliasing in magnified and minified regions.

2. Related Work

In this section we briefly review recent approaches to hardware-accelerated surface splatting. For a more detailed discussion on point-based rendering the reader is referred to the surveys [KB04, SP04]. We roughly classify the methods discussed below by the compromises they had to make due to GPU limitations, or, equivalently, by the amount of programmable shader features they exploit.

All approaches have in common that they use three rendering passes in order to achieve correct blending of overlapping splats. In the first pass called *visibility splatting* the object is rendered without lighting in order to fill the depth buffer only [RL00]. In the *blending* pass the object is slightly shifted towards the viewer by ε and rendered with lighting and alpha blending options enabled. This achieves a Gouraud-like blending of overlapping splats whose depths differ by less than ε, but still leads to correct occlusions for splats with larger depth offsets. In a final *normalization* pass each pixel is normalized by dividing the accumulated colors stored in its RGB components by the sum of weights stored in its alpha component, which can be done directly on the GPU [BK03, GP03].

Early methods do not exploit pixel shaders for the splat rasterization, but instead represent each splat by an alpha-textured quad [RPZ02] or triangle [PSG04]. However, this multiplies the memory consumption as well as the per-vertex computation costs by a factor of 4 or 3, respectively, which is especially critical for highly complex or dynamically changing geometries.

This overhead can be avoided by representing each splat by just one OpenGL vertex and using pixel shaders for their rasterization instead [BK03, GP03]. In this case, special care has to be taken to generate correct depth values for each pixel, since this is a requirement for correct blending results. However, both methods only approximate the exact elliptical shape of the projected splats, which might lead to small holes in the resulting image. These holes can be avoided either by a perspectively more accurate affine approximation that correctly maps the splat contours [ZRB*04], or by a per-pixel projectively correct ray casting approach [BSK04].

Sharp surface features can be represented by clipping splats at lines defined in their tangent space, which was first proposed by [PKKG03]. Efficient hardware implementations of this technique were then presented in [ZRB*04, BSK04]. Since these methods cut splats by boolean intersections with tangent half-spaces, the resulting clipped splats are always convex. For more complicated sharp features one therefore has to fall back to a software-based rendering solution [WTG04].

In comparison to Gouraud-like shading by blending the colors of overlapping splats, per-pixel shading considerably improves the visual quality [ZPvBG01, KV01]. The GPU-accelerated Phong splatting approach [BSK04] uses the original point normals for precomputing a linear normal field for each splat, which is evaluated at render-time for per-pixel lighting computations. In combination with the splat decimation technique of [WK04] they achieve high rendering quality even for coarsely sampled models. However, as the normal fields have to be precomputed, their approach would not be suitable for dynamically changing geometries, like for instance [BK05]. Since this approach requires to transfer more data to the pixels shaders, and since the normal field evaluation and per-pixel lighting complicate the computations, this approach is limited to about 6M splats/sec,

© The Eurographics Association 2005.

measured on a Linux machine equipped with a 3.0GHz Pentium4 CPU and a NVIDIA GeForce 6800 Ultra GPU. All timings and splat rates given in this paper were measured using this configuration.

Most of the above methods neglect the screen space filter of the EWA framework and restrict to the Gaussian reconstruction filter in object space. While this leads to sufficient anti-aliasing in magnified regions, it cannot prevent aliasing artifacts in minified areas. In contrast, the method of [ZRB*04] implements the full EWA splatting approach on the GPU, but due to the complex computations its performance is limited to about 4M splats/sec using the above mentioned hardware configuration. Finally, the use of deferred shading techniques has shown to allow for efficient high-quality per-pixel shading in EWA surface splatting as well as splat-based volume rendering [MMC99].

Using per-pixel Phong shading and a simple but effective approximation to the screen space filter, the approach presented in this paper provides results comparable to the original EWA splatting. By exploiting deferred shading techniques we achieve this superior visual quality at a rate of about 23M splats/sec, such that our method is close to EWA splatting in terms of quality and close to the fast but low-quality renderer of [BK03], which achieves about 27M splats/sec.

3. Splat Rasterization

In this section we shortly describe the perspectively correct rasterization of elliptical splats, which was introduced in [BSK04]. This method will then be used in the next section to accumulate normal and color contributions of individual splats in the rendering buffer.

Following the notation of [BSK04], a splat S_j is defined by its center $\mathbf{c}_j$ and two orthogonal tangent directions $\mathbf{u}_j$ and $\mathbf{v}_j$. These tangent vectors are scaled according to the principal radii of the elliptical splat, such that an arbitrary point $\mathbf{q}$ in the splat's embedding plane lies in the interior of the splat if its local parameter values u and v satisfy the condition

$$u^2+v^2 \;=\; \Big(\mathbf{u}_j^T\,(\mathbf{q}-\mathbf{c}_j)\Big)^2 + \Big(\mathbf{v}_j^T\,(\mathbf{q}-\mathbf{c}_j)\Big)^2 \;\leq\; 1 \quad . \qquad (1)$$

The rasterization of a splat S_j is performed by sending its center $\mathbf{c}_j$, tangent axes $(\mathbf{u}_j,\mathbf{v}_j)$, and optional material properties to OpenGL, which are then processed by custom shaders for both the vertex and the pixel stage. The vertex shader conservatively estimates the size d of the projected splat based on a perspective division of the larger of the ellipse radii r by the eye-space depth value $\mathbf{c}_z$ of the splat center, followed by a window-to-viewport scaling as described in [BSK04].

This causes the single OpenGL vertex $\mathbf{c}$ to be rasterized as a $d \times d$ image space square, each pixel (x,y) of which is then tested by a pixel shader to lie either inside or outside of the projected elliptical splat contour. Local ray casting through the corresponding projected point $\mathbf{q}_n$ on the near plane yields the eye-space point $\mathbf{q}$ on the splat's supporting plane. From this projectively exact 3D position the local parameter values (u,v) can be determined and tested as shown in (1). While pixels corresponding to points outside the splat are discarded, pixels belonging to the splat are accepted and processed further. If a pixel (x,y) is accepted, its weighting factor is determined as

$$w(x,y) \;=\; h\left(\sqrt{u^2+v^2}\right) \quad , \qquad (2)$$

where $h(\cdot)$ is typically chosen as a Gaussian. To allow for exact blending and occlusion, the pixel's depth value has to be adjusted as described in [BSK04] in order to correspond to the computed 3D position $\mathbf{q}$. This finally results in a per-pixel projectively correct rasterization of elliptical splats.

The output of the rasterization pixel shader are depth values only for the visibility pass, and additionally weighted splat attributes, such as normal vectors or color values, in the second pass, which are then accumulated in the render target by additive alpha blending. The final normalization/shading pass then processes each pixel in order to compute its final color, as described in the next section.

4. Hardware-Accelerated Deferred Shading

For both point-based models and polygonal meshes, one major requirement for high-quality visualization is the use of per-pixel Phong shading based on interpolated normal vectors, instead of Gouraud shading, which blends color contributions resulting from lighting each splat or mesh vertex, respectively. In contrast to polygonal meshes, point-based models do not store any neighborhood relation between splats, therefore an equivalent interpolation of neighboring splats' normal vectors is not possible.

In order to still be able to generate smoothly interpolated per-pixel normal vectors, two basic approaches are possible. The first is to associate with each splat a pre-computed linear normal field, as proposed in the Phong splatting approach [BSK04]. However, while leading to a high-quality shading, this method is limited to static geometries and bound to about 6M splats/sec, as mentioned in Section 2.

The second approach for normal interpolation was proposed in the software-based EWA splatting approach [ZPvBG01]. Instead of splatting color values into the framebuffer, they use multiple buffers into which they splat normal vectors and material properties. As a consequence, normals and colors of overlapping splats are smoothly interpolated and averaged into the pixels they cover, with weights depending on the respective EWA filter kernels evaluated at that pixel. In a final pass over each image pixel, lighting computations are performed based on the pixel's accumulated normal vector and surface material.

© The Eurographics Association 2005.

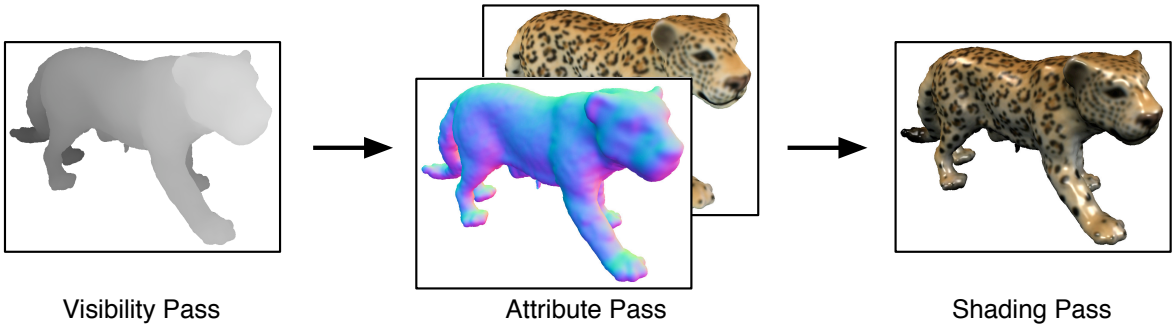

Figure 1: *The deferred shading pipeline for GPU-based splatting. The* visibility *pass fills the z-buffer, such that the* attribute *pass can correctly accumulate surface attributes, like color values and normal vectors, in separate render targets. The final* shading *pass computes the actual color value for each image pixel based on the information stored in these render targets.*

4.1. Multipass Algorithm

The latest NV40 generation of NVIDIA GPUs provides all the hardware features required to implement the latter approach on the GPU. Originally targeted at high-quality cinematic rendering effects and high dynamic range imaging, the NV40 provides floating point precision at all necessary stages of the rendering pipeline, i.e., for shader arithmetic, alpha blending, textures, and render targets. In combination with the availability of multiple render targets, which allow outputting up to four different RGBA color values within a single rendering pass, these features enable the implementation of accurate per-pixel deferred shading in the context of surface splatting.

Attribute Pass. After the visibility pass (cf. Fig. 1, left), we use multiple render targets to splat and accumulate normal vectors as well as material properties during the so-called attribute pass (cf. Fig. 1, center). The corresponding pixel shader performs the computations outlined in Section 3, but instead of shading each accepted pixel, its (weighted) normal vector and color value are output to the two render targets. These buffers and the depth buffer are then used as textures for the final normalization and shading pass, for which a window-size rectangle is drawn in order to send each pixel through the rendering pipeline again.

Shading Pass. The shading pass (cf. Fig. 1, right) corresponds to the normalization pass of previous approaches, but it additionally performs (deferred) per-pixel shading. For each pixel, an averaged normal and color can be computed by fetching the accumulated values from the textures and normalizing them. From the depth texture, the corresponding 3D position can easily be derived by inverting the viewing and projection mappings. Having position, normal, and color information at hand then enables *deferred* per-pixel shading computations [DWS*88]. The resulting Phong shading clearly improves the rendering quality over the Gouraud shading used by most previous methods.

Notice that lighting computations are performed only once for each pixel of the projected object in the final image. In contrast, previous approaches incorporate lighting computations into the splat rasterization process and perform a per-pixel blending of the resulting colors instead. Due to the required mutual overlap of individual splats, this multiplies the number of lighting computations by a factor of about 6–10 for typical datasets, which we measured by counting the fragments contributing to each pixel using the stencil buffer.

Depending on the complexity of the employed shaders, saving these unnecessary lighting computations yields noticeable performance improvements. As we will show in Section 6, the performance of our deferred shading approach is almost independent of the actual surface shading. Incorporating more complex lighting computations into the rasterization pixel shader would in contrast significantly slow down the rendering, as the pixel stage is known to be the bottleneck of the splat rasterization.

In addition to this, deferred shading also provides a clear separation between the splat rasterization process and the actual surface lighting or shading computations. This greatly simplifies the development of custom shaders, as the carefully optimized pixel shader for splat rasterization (cf. Section 3) is left untouched. The deferred shading approach thus allows for a simple yet highly efficient implementation of custom shaders, of which we show several examples in Section 6. Since the input to these shaders are textures holding normal, material, and depth information, they are independent of the actual geometry that was rasterized to generate these textures. As a consequence, the shaders can even be shared for point-based models and traditional polygonal meshes.

© The Eurographics Association 2005.

Another important point to be considered is the precision of the render targets. The standard framebuffer used in previous approaches offers 8 bits for each of the four RGBA components. As an additional constraint, these color channel also have to be clamped to $[0,1]$. This leads to the frequently observed shading artifacts due to color buffer overflows or insufficient precision for the sum of weights stored in the alpha channel. The NV40 GPU generation now allows to use un-clamped floating point values for render targets, which effectively avoids these problems (cf. Fig. 2). This is especially important as in addition to colors we also accumulate normal vectors, where noise due to discretization would immediately lead to shading artifacts.

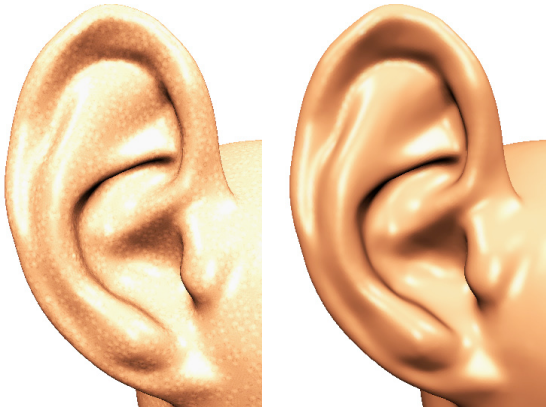

Figure 2: *Standard framebuffers provide 8 bit precision for each channel and clamp color values to* $[0,1]$*. Due to large overlaps of individual splats, these buffers may overflow during accumulation, resulting in the too bright and sparkled* left *image. Using floating point render targets (on the same illumination conditions) effectively avoids these problems (*right*).*

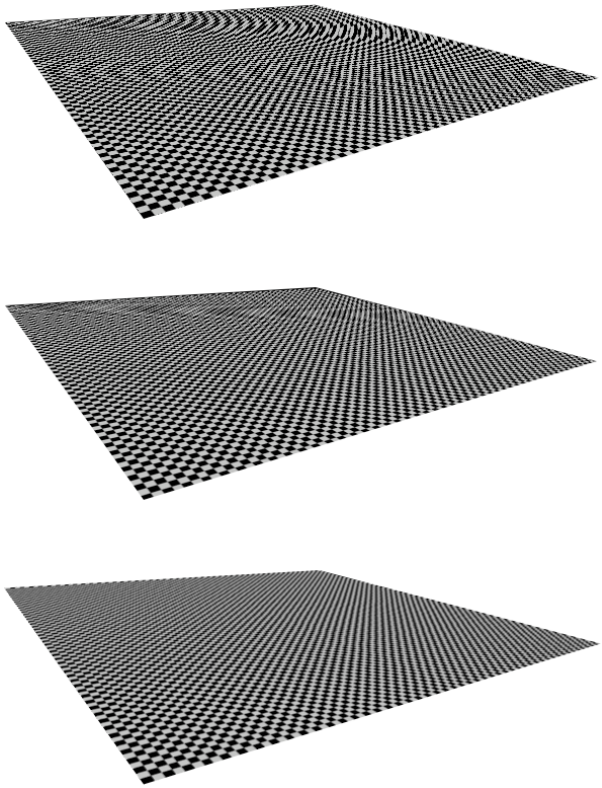

Figure 3: *The object-space reconstruction filter alone cannot avoid aliasing in minification regions (*top*). Full-screen anti-aliasing removes aliasing to some degree, but the super-sampled image can still contain sampling artifacts (*center*). Our approximation to the EWA filter band-limits the signal* before *it is sampled on the pixel grid and hence successfully removes the aliasing problems (*bottom*).*

5. EWA Approximation

In the original EWA surface splatting, two components are responsible for high visual quality: per-pixel Phong shading, which can be mapped to the GPU as shown in the last section, and anisotropic anti-aliasing provided by the EWA filter.

The complete EWA filter is composed of an object-space reconstruction kernel (the weight function of Eq. (2)) and a band-limiting screen-space pre-filter. As the required computations are quite involved, many rendering approaches simply omit the screen-space filter and use the reconstruction kernel only. However, in the case of extreme minification, when the size of projected splats falls below one pixel, the signal corresponding to the accumulated projected splats may have frequencies higher than the Nyquist frequency of the pixel sampling grid, resulting in the alias artifacts shown in the top image of Fig. 3.

An appealing idea might be to diminish these aliasing artifacts by full-screen anti-aliasing (FSAA), which is supported by any modern graphics hardware. In general, FSAA redirects the rendering to a higher resolution framebuffer in order to achieve a (typically 2×2 or 3×3) super-sampling of the image signal. This buffer is then scaled down to the actual framebuffer resolution using linear or Gaussian filtering. The problem with that approach is that even the high resolution super-sampling buffer might suffer from aliasing, in which case a high resolution *aliased* image will be downscaled to the framebuffer. The resulting image will still contain alias artifacts (cf. Fig. 3, center).

We propose a simple — and hence efficient — heuristic for approximating the EWA screen-space filter. By clamping the size of projected splats to be at least 2×2 pixels it is guaranteed that enough fragments are generated for anti-aliasing purposes, even for splats projecting to sub-pixel ar-

© The Eurographics Association 2005.

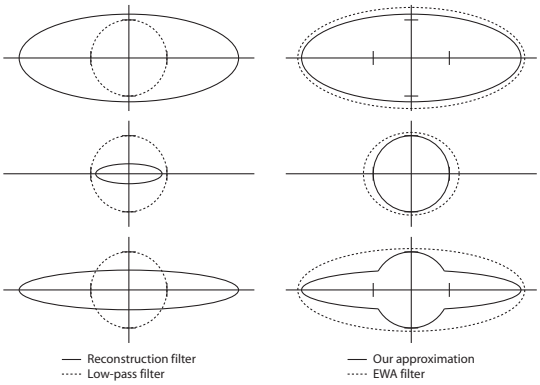

Figure 4: *This figure gives a qualitative comparison of the original EWA filter and our approximation. In the left column, three typical configurations of screen-size ratios between the projected object-space reconstruction filter and the low-pass screen-space filter are shown. The right column compares the resulting contours of the combined filter kernels. Although the approximation error can become arbitrarily large, we did not perceive any visible artifacts in our experiments.*

eas. This restriction on the minimum size can easily be incorporated into the vertex shader.

Instead of computing the weight $w(x,y)$ based only on the reconstruction filter, the pixel shader is adjusted to compute two radii $r_{3D} := u^2 + v^2$ (see Eq. (1)) and $r_{2D} := d(x,y)^2 / r^2$, with $d(x,y)$ being the 2D distance of the current fragment from the respective projected splat center and $r = \sqrt{2}$ being the band-limiting screen-space filter radius. A given fragment is then accepted, if it lies within the union of the low-pass and the reconstruction filter (cf. Fig. 4)

$$\tilde{r}(x,y) \;:=\; \min\{r_{2D}(x,y),\, r_{3D}(x,y)\} \leq 1 \quad ,$$

i.e., either if it corresponds to a 3D point within the splat's interior, or if it lies within a certain radius around the projected spat center. The final weight corresponding to Eq. (2) is computed as $w(x,y) = h(\sqrt{\tilde{r}(x,y)})$.

Notice that we enforce the minimal splat size only in the attribute pass, but not in the visibility pass. This means that the ε-depth test, which is simulated by the two rendering passes, is not applied to those pixels which are additionally generated on silhouettes by the screen-space filter. In contrast, these pixels are blended with the surface parts behind them, which results in a pseudo edge-anti-aliasing for object silhouettes.

This approximation to the EWA filter provides high-quality anti-aliasing in magnified as well as in minified regions (cf. Fig. 3, bottom). Our results are comparable to those of the exact EWA filter, but in contrast our approximation is considerably easier to compute. If the projected splat center is passed from the vertex shader to the pixel shader, the screen-space filter requires three additional instructions only.

Limiting the minimal projected splat size obviously generates more fragments, such that the average number of fragments contributing to each resulting image pixel increases by a maximum factor of 4 from about 7 to 15–30 for complex models with small projected splat sizes. As a consequence, the acceleration offered by the deferred shading approach becomes even more important, since by this the screen-space filter decreases the rendering performance only slightly from 25M to 23M splats/sec.

6. Results

In this section we discuss the quality and efficiency of the presented approach and compare both criteria to previous GPU-based renderers. The clear separation of splat rasterization and (deferred) surface shading allows for easy and efficient implementation of custom shading methods, for which we show several examples.

In terms of quality, our method can be compared to the software-based EWA splatting [ZPvBG01] and the GPU-based Phong splatting [BSK04]. Our per-pixel Phong shading is a GPU implementation of the original EWA splatting and therefore yields equivalent results. However, our splat rasterization is perspectively correct, whereas the affine approximation of the projective mapping in [ZPvBG01] might cause small holes in the image, as pointed out in [ZRB*04].

In comparison to Phong splatting, our per-pixel shading provides equivalent results for densely sampled models. For coarse models, Phong splatting can achieve better results by pre-computing the linear normal fields from the undecimated original dense models. However, as Phong splatting does not use a screen-space pre-filter, this method might suffer from aliasing in minification areas.

In order to test the performance implications of the surface shader's complexity, we compared the simple Phong shader using a precomputed light map [BSK04] to a more complex non-photo-realistic (NPR) shader (cf. Fig. 5). The latter generates contour and silhouette outlines at depth continuities and at shallow viewing angles. Using deferred shading, this technique can be implemented easily by sampling the depth map in the pixel's neighborhood and by considering the normal map information. The diffuse lighting result is further modified by a 1D transfer function texture, e.g., in order to achieve the quantized color set typical for toon shading [GGC98]. In total, the NPR shader requires 12 texture lookups and is much more involved compared to the Phong shader, which needs only 3 texture fetches. However, our results show that, due to deferred shading, the effect on the total timings are almost negligible, whereas for non-deferred splatting, the shader complexity has a considerably larger impact on the rendering speed.

© The Eurographics Association 2005.

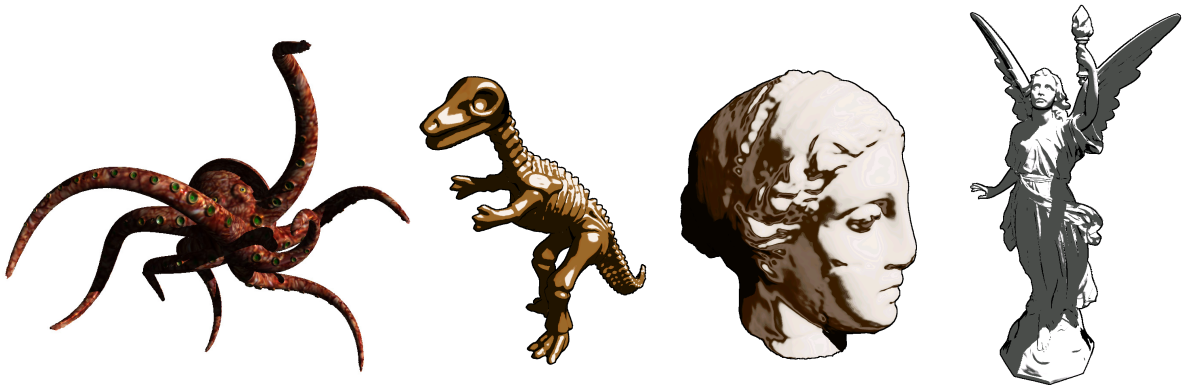

Figure 5: *From left to right: The Phong-shaded octopus model and NPR-shaded renderings of the dinosaur model, the Igea artifact, and the massive Lucy dataset. All models are rendered with shadow mapping enabled and hence require one additional visibility rendering pass for the shadow map generation.*

We further enhanced the Phong shader as well as the NPR shader by standard shadow mapping, which requires transforming each image pixel first back to object-space and then into the viewport coordinate system of the shadow map. Based on the comparison of the resulting depth value with the corresponding entry of the shadow map, lighting is performed only in un-shadowed regions. The additional shader computations again have a minor influence on the rendering performance, but as one extra visibility pass is required for the generation of the shadow map, the splat rate drops down to about 15M splats/sec.

Table 1 shows performance statistics of our Phong and NPR shader, both with and without shadow mapping. The numbers correspond to splat rates in million splats per second, for a 512×512 window, using a GeForce 6800 Ultra GPU. We also give the average number of fragments contributing to each final image pixel in order to show the amount of shading computations which is saved by the deferred shading approach. A comparison to the high performance but low quality renderer of [BK03] and the high-quality but low-performance Phong splatting [BSK04] on identical hardware reveals that our method achieves a performance close to the fast low-quality renderer, but yields a quality superior to even the Phong splatting approach.

In order to minimize data transfer costs, the geometry data is stored in high performance GPU memory using OpenGL vertex buffer objects. For massive models, however, this data might not fit into the available GPU memory. In order to still be able to efficiently render those datasets, we switch to 16 bit floating point values for representing positions and normals. This effectively halves the memory consumption without leading to any noticeable loss of visible quality. The two most complex models shown in Table 1 still cannot be uploaded to the video memory, but as the data transfer costs are also halved by the 16 bit quantization, they can be rendered quite efficiently. The massive Lucy model can therefore be rendered at 1.6 fps with Phong shading and approximate EWA anti-aliasing, and at still more than 1 fps with the 4-pass shadow mapping shaders.

7. Conclusion

In this paper we showed how to exploit the increased capabilities of the latest generation of graphics processors for point-based rendering. The availability of multiple render targets in combination with a floating point precision rendering pipeline enabled us to derive one of the fastest and highest quality GPU-based surface splatting technique available to date.

This was achieved by introducing a splat-rendering pipeline based on deferred shading and an approximation to the screen-space EWA filter. Due to the required mutual overlap of individual surface splats, deferred shading was shown to be especially suited for point-based rendering, and to provide high-quality per-pixel shading as well as significant performance improvements. Our approximation to the EWA screen-space filter effectively removes aliasing artifacts in minified areas, while still preserving the superior rendering performance.

The most problematic limitation for current point-based rendering approaches is still the restricted flexibility of z-buffering, which necessitates the expensive two render passes for visibility splatting and attribute blending, and which also restricts current surface splatting approaches to completely opaque surfaces only.

Acknowledgments

The octopus model is courtesy of Mark Pauly, the Lucy model courtesy of Stanford university, and the dinosaur and Igea datasets are courtesy of Cyberware.

© The Eurographics Association 2005.

Model	#splats	Overdraw	Phong-SM	NPR-SM	Phong+SM	NPR+SM	[BK03]	[BSK04]
Balljoint	137k	5.9 / 7.2	20.1	15.5	13.3	11.9	24.8	4.5
Max	655k	7.9 / 15.4	22.0	20.4	14.9	14.5	27.0	5.9
David Head	1.1M	6.4 / 14.4	23.9	22.6	16.3	15.8	27.1	5.6
David Head	4.0M	6.9 / 37.7	26.0	25.3	17.5	17.2	31.3	5.5
David	8.3M	7.0 / 202	22.6	22.0	16.2	15.5	19.6	4.6
Lucy	14M	19.3 / 242	22.6	22.1	15.9	15.2	20.2	5.0

Table 1: *This table shows the performance of our rendering approach in million splats per second for a* 512×512 *window, using a GeForce 6800 Ultra GPU. We give timings for several different shaders (Phong shading, NPR shading, with and without shadow mapping) and compare to the fast but low-quality splatting of [BK03], and the high-quality but expensive Phong splatting [BSK04]. Due to per-pixel Phong shading and anti-aliasing, the quality of our method is superior even to [BSK04], while the rendering performance is still comparable to [BK03]. The third column shows the average number of fragments contributing to resulting image pixels, without and with our anti-aliasing technique. Since the latter generates significantly more fragments for complex models, the acceleration provided by the deferred shading approach is even more important.*

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proc. of IEEE Visualization 01* (2001), pp. 21–28.

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

[BK05] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. In *Proc. of Eurographics 05* (2005).

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Proc. of symposium on Point-Based Graphics 04* (2004).

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *Proc. of ACM SIGGRAPH 88* (1988), pp. 21–30.

[GGC98] GOOCH A., GOOCH B., COHEN E.: A non-photorealistic lighting model for automatic technical illustration. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 447–452.

[GP03] GUENNEBAUD G., PAULIN M.: Efficient screen space approach for hardware accelerated surfel rendering. In *Proc. of Vision, Modeling, and Visualization 03* (2003).

[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 *Proc. of Eurographics Workshop on Rendering Techniques 2001* (2001).

[MMC99] MUELLER K., MÖLLER T., CRAWFIS R.: Splatting without the blur. In *Proc. of IEEE Visualization* (1999), pp. 363–370.

[PKKG03] PAULY M., KEISER R., KOBBELT L., GROSS M.: Shape modeling with point-sampled geometry. In *Proc. of ACM SIGGRAPH 03* (2003), pp. 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 (2004), 598–608.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 335–342.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: a multiresolution point rendering system for large meshes. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 343–352.

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

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

[WK04] WU J., KOBBELT L.: Optimized subsampling of point sets for surface splatting. In *Proc. of Eurographics 04* (2004), pp. 643–652.

[WTG04] WICKE M., TESCHNER M., GROSS M.: CSG tree rendering for point-sampled objects. In *Proc. of Pacific Graphics 04* (2004), pp. 160–168.

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. of ACM SIGGRAPH 01* (2001), pp. 371–378.

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

© The Eurographics Association 2005.