Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa, S. Rusinkiewicz, (Editors)

Phong Splatting

Mario Botsch Michael Spernat Leif Kobbelt

Computer Graphics Group
RWTH Aachen University

Abstract

Surface splatting has developed into a valuable alternative to triangle meshes when it comes to rendering of highly detailed massive datasets. However, even highly accurate splat approximations of the given geometry may sometimes not provide a sufficient rendering quality since surface lighting mostly depends on normal vectors whose deviation is not bounded by the Hausdorff approximation error. Moreover, current point-based rendering systems usually associate a constant normal vector with each splat, leading to rendering results which are comparable to flat or Gouraud shading for polygon meshes.
In contrast, we propose to base the lighting of a splat on a linearly varying normal field associated with it, and we show that the resulting Phong Splats *provide a visual quality which is far superior to existing approaches. We present a simple and effective way to construct a Phong splat representation for a given set of input samples.*
Our surface splatting system is implemented completely based on vertex and pixel shaders of current GPUs and achieves a splat rate of up to 4M Phong shaded, filtered, and blended splats per second. In contrast to previous work, our scan conversion is projectively correct per pixel, leading to more accurate visualization and clipping at sharp features.

1. Introduction

Surface splatting is a well-established technique to render high quality images of geometric objects that are given by a sufficiently dense set of sample points. The idea is to approximate local regions of the surface by planar ellipses in object space and then render the surface by accumulating and blending these ellipses in image space. From the geometric point of view, the set of ellipses defines a piecewise linear approximation of the given geometry, and the size and aspect ratio of the ellipses depend on the local principal curvatures (and the approximation tolerance prescribed by the user).

Usually, each splat is associated with a normal vector in order to compute local lighting, which leads to a piecewise constant shading — similar to flat shading for polygonal meshes. Blending the splats' color contributions by Gaussian filtering improves the rendering from flat shading to Gouraud shading. For polygons, much better visual quality is achieved by *Phong shading*, where normal vectors are linearly interpolated across triangles, leading to a continuously varying lighting [Bui75]. However, normal interpolation requires connectivity information which is usually not available for surface splats.

The basic idea of *Phong splatting* is to associate a linearly varying normal field with each splat instead of keeping the normal constant. By this, we achieve the same visual quality as Phong shaded polygons, but we preserve all the important advantages of point-based surface representations, e.g., we do not have to construct a globally consistent connectivity.

The first to propose the use of a varying normal field for surface splatting were [KV01], generating normal mapped splats by sampling position and curvature information of NURBS surfaces. In this paper we present an algorithm for the generation of Phong splats that provides a simpler formulation as well as a more robust and accurate approximation of the geometry as well as of its normal field.

In order to generate Phong splat representations we assume that the input data consists of a set of sample points with associated normal vectors. Hence, each point can be considered as a 5-dimensional sample. The first three entries are the spatial point coordinates and the last two entries represent the (normalized) normal vector.

To approximate a certain surface region by an elliptic splat, we fit a least squares plane to the corresponding set of sample points that belong to this region. Then we project the samples to this plane and fit a minimum enclosing ellipse. To derive the linear normal field, we do exactly the same, i.e. we fit a linear function to the normal vectors in the least squares sense.

The rendering of the resulting Phong splat representations can be implemented very efficiently by exploiting the features of modern programmable graphic processors. Our system delegates all rendering tasks to the vertex and pixel shader units of the GPU and achieves a rendering speed of up to 4M high-quality filtered Phong splats per second. Compared to standard point-based rendering algorithms, Phong splatting needs significantly fewer splat primitives to produce the same visual quality. Taking this into account, our method is also more efficient than previous high-quality splatting approaches.

Besides higher performance, another advantage of mapping the complete rendering process to the GPU is that the CPU is free to handle the actual processing of the geometry data. Hence, the geometry processing algorithms also benefit from Phong splats, as less primitives have to be stored and processed.

2. Related Work

Points have first been proposed as rendering primitives by Levoy and Whitted [LW85], followed by point-based rendering approaches based on image-space reconstruction techniques [GD98, PZvBG00, SD01] or object-space resampling [ABCO*01, FCOAS03].

In contrast to this, surface splatting [ZPvBG01] avoids holes in the rendered image by associating a normal vector and a radius with each point, thereby considering them as small discs or ellipses in objects space. Building on this work, several point-based rendering approaches have been developed, the early ones being implemented in software and therefore putting a high load on the CPU [ZPvBG01, BWK02].

The increasing efficiency and programmability of modern graphic cards triggered the development of hardware-based splatting methods, starting with [RL00]. The approach of [RPZ02] renders splats using object space quads, causing a multiplication of the number of vertices to be processed by a factor of four. More efficient approaches manage to render just one vertex per splat and use vertex and pixel shaders for the splat rasterization. But also these methods represent a trade-off between high visual quality and efficient rendering, differing in the rendering primitives and filtering methods they use. Splat shapes range from simple image space squares [DVS03] over circular splats [CH02, BK03] to elliptical splats that nicely adapt to the local surface curvature [ZRB*04].

High quality anisotropic anti-aliasing can be achieved by assigning a Gaussian filter kernel to splats. Combining these object-space reconstruction kernels with a band-limiting image-space filter results in the high quality EWA splatting technique [ZPvBG01, RPZ02, ZRB*04].

However, all methods mentioned above use a constant normal vector for lighting splats, leading to results similar to flat shading in the polygon rendering case. Blurring the shading discontinuities by Gaussian filtering achieves renderings comparable to Gouraud shading. Better results can be achieved by a per-pixel EWA averaging of normals in combination with deferred shading, as proposed by [ZPvBG01]. However, non-constant normal fields yield an even higher visual quality, as first proposed by Kalaiah and Varshney [KV01, KV03]. Since their approach is the one most similar to ours, we will compare to it in more detail in Sec. 5. A comparison to [ZPvBG01] is given in Sec. 6.

Being piecewise linear primitives, surface splats also gained increasing attention as an alternative geometry representation, since they exhibit the same approximation power as triangle meshes. As a consequence, several geometry processing algorithms have been developed for point-based geometries [PG01, ZPKG02, PKKG03, PKG03]. The running time of these algorithms is mostly dominated by the number of splat primitives. To enable efficient processing and rendering of these datasets, simplification and resampling techniques for point-based geometry have been developed [PGK02, Paj03, WK04]. We build upon this kind of approaches by adding a linearly varying normal field to the splat primitives.

We simplify and improve the splat generation phase of [KV03] in Sec. 3 and propose a Phong splat rendering system that combines per-pixel lighting and high-quality filtering and correctly computes the projection per pixel in Sec. 4. Finally Sec. 6 shows that the resulting Phong splatting framework provides a superior ratio of quality to speed compared to existing splatting approaches.

3. Phong Splat Generation

Given a dense set of (input) surface samples $\mathbf{p}_i$ we use a technique like the one described in [WK04] to generate an optimized set of elliptical surface splats that approximates the input data within a prescribed error tolerance ε. The idea of this technique is to locally estimate the principal curvature directions at every input sample and to align the ellipses' axes accordingly. An initial selection of splats that cover the complete set of samples is then optimized by a global relaxation procedure.

The output of this procedure is a set of elliptical surface splats S_j that provide an approximation of the input point cloud. A splat is defined by its center $\mathbf{c}_j$ and two orthogonal principal tangent directions $\mathbf{u}_j$ and $\mathbf{v}_j$. The tangents are scaled according to the corresponding ellipse radii such that

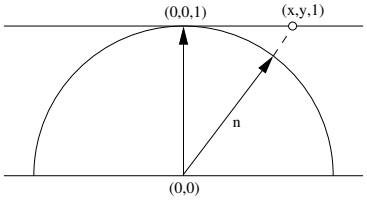

Figure 1: *Since lengths are not important for the normal fitting procedure, normals are represented as homogeneous points on an offset tangent plane.*

an arbitrary point $\mathbf{q}$ in the plane spanned by $\mathbf{u}_j$ and $\mathbf{v}_j$ lies in the interior of the splat S_j if its local parameter values u and v satisfy the condition

$$u^2 + v^2 = \left(\mathbf{u}_j^T(\mathbf{q}-\mathbf{c}_j)\right)^2 + \left(\mathbf{v}_j^T(\mathbf{q}-\mathbf{c}_j)\right)^2 \leq 1. \quad (1)$$

The inside test can be applied even to points that do not lie in the supporting plane of the splat. In this case, the two parameter values u and v correspond to the orthogonal projection of $\mathbf{q}$ into that plane.

Each splat S_j is associated with a sub-set P_j of the samples $\mathbf{p}_i$ which are covered by it, i.e., which are contained in the elliptical cylinder generated by offsetting the splat in normal direction by ε and by $-\varepsilon$ respectively. Notice that usually we expect $P_j \cap P_k \neq \emptyset$ for neighboring splats due to the mutual overlap.

The basic idea of *Phong shading* is to assign explicit normal vectors to the vertices of a polygon mesh and then to interpolate these normals in a piecewise linear fashion. *Phong splatting* does the same, except that we cannot interpolate normal vectors of neighboring splats, since in general no connectivity information is available. Hence, for each splat we derive a linear normal field from the associated set of samples P_j.

For the geometric fitting the splat's center and tangential directions are usually derived by fitting a plane to the sample points $\mathbf{p}_i \in P_j$ in the least squares sense. We do exactly the same in order to fit a linear normal field to the given normals $\mathbf{n}_i$ associated with the sample points $\mathbf{p}_i$. This normal field N_j is specified by a center normal $\bar{\mathbf{n}}_j$ and two scalar values α_j and β_j, such that the (unnormalized) normal of a point $\mathbf{q}$ on the splat S_j with parameter values (u,v) is

$$N_j(u,v) = \bar{\mathbf{n}}_j + u\,\alpha_j\,\mathbf{u}_j + v\,\beta_j\,\mathbf{v}_j, \quad (2)$$

i.e. we tilt the center normal along the tangential directions.

For the normal fitting we represent a given normal vector $\mathbf{n}_i$ w.r.t. the *local frame* spanned by the splat's tangent directions $(\mathbf{u}_j, \mathbf{v}_j)$ and its normal $(\mathbf{u}_j \times \mathbf{v}_j)$. Note that the center normal $\bar{\mathbf{n}}_j$ generally differs from $\mathbf{u}_j \times \mathbf{v}_j$ in order to minimize the normal error over the whole splat.

For later lighting computations the length of the interpolated normal vector is not relevant, since it can be re-normalized or is used for accessing a cube-map (Sec. 4). As a consequence, we can set the third local-frame coordinate of $\mathbf{n}_i$ to 1 such that each normal vector is actually represented by a point (x,y) on an offset tangent plane, similar to homogeneous coordinates (cf. Fig. 1).

If the center normal $\bar{\mathbf{n}}$ is represented by $(\bar{x},\bar{y})$, and if (u_i,v_i) denote the parameter values of the sample $\mathbf{p_i}$ and (x_i,y_i) its local-frame normal vector, the normal fitting can be written as a set of linear equations

$$\begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} + \begin{pmatrix} u_i\,\alpha \\ v_i\,\beta \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \forall\, \mathbf{p}_i \in P_j,$$

which are solved for $(\bar{x},\bar{y})$, α and β in the least squares sense, e.g. using the normal equations $(A^TAx = A^Tb)$. Since in the above equation the x and y components are uncoupled, it can be further simplified to the solution of two 2×2 linear systems:

$$\begin{pmatrix} |P_j| & \sum u_i \\ \sum u_i & \sum u_i^2 \end{pmatrix} \begin{pmatrix} \bar{x} \\ \alpha \end{pmatrix} = \begin{pmatrix} \sum x_i \\ \sum x_i\,u_i \end{pmatrix} \quad (3)$$

$$\begin{pmatrix} |P_j| & \sum v_i \\ \sum v_i & \sum v_i^2 \end{pmatrix} \begin{pmatrix} \bar{y} \\ \beta \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum y_i\,v_i \end{pmatrix}, \quad (4)$$

where the summation is done over all $\mathbf{p}_i \in P_j$. If these systems happen to be underdetermined, e.g. if P_j contains less than 2 samples, we compute the least norm solution using the pseudo-inverse [GL89].

Note that the overall method works best if the tangential directions are roughly aligned to the directions of minimal and maximum normal deviation of the sample normals $\mathbf{n}_i$, i.e. to the principal curvature directions. If this is not already provided by the geometry fitting scheme, like e.g. [WK04], the directions $\mathbf{u}_j$ and $\mathbf{v}_j$ can easily be estimated by the eigenvectors corresponding to the two smaller eigenvalues of the covariance matrix of sample normals $\sum_i \mathbf{n}_i\mathbf{n}_i^T$ (see [Gar99] for details).

The result of this normal fitting process is a center normal $\bar{\mathbf{n}}_j$ and two scalars α_j and β_j associated with each splat S_j, representing an optimal fit to the given input normals.

4. Phong Splat Rendering

In this section we show how to render the Phong splat representation based on the features of modern GPUs. The Phong splatting system we propose is targeting at superior visual quality, therefore we combine per-pixel lighting using linear normal fields with elliptical splat shapes and antialiasing by Gaussian filtering and blending.

For efficiency reasons we want to render each splat by sending only one vertex through the graphics pipeline. Like in previous approaches, we determine the projected size of the splat and adjust the OpenGL point size, such that a sufficiently large square will be generated and rasterized. In order

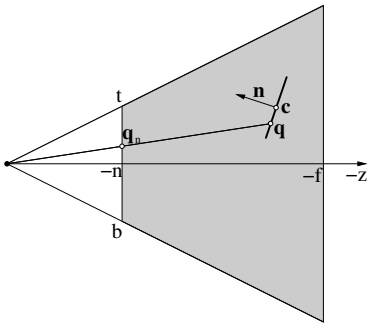

Figure 2: *We compute the point* $\mathbf{q}$ *corresponding to a given window pixel by casting a ray through the respective point* $\mathbf{q}_n$ *on the near plane and intersecting it with the splat plane.*

to avoid the complicated computation of the exact projected size [ZRB*04], we conservatively estimate it by perspectively foreshortening the larger of the ellipse radii r using the depth value z_{eye} of the splat center $\mathbf{c}$ like in [BK03]

$$size = 2r \cdot \frac{n}{z_{eye}} \cdot \frac{h_{vp}}{t-b}, \tag{5}$$

where n, t and b are the parameters of the viewing frustum (cf. Fig. 2) and h_{vp} is the height of the viewport.

This will rasterize a $size \times size$ image-space square such that we have to determine the local splat parameter values for each pixel generated by it. Based on these coordinates we decide whether the pixel is within the splat or should be discarded, leading to the correct elliptical splat shape. In contrast to other approaches we base this on the computation of the exact 3D point corresponding to the current pixel position by inverting the viewing and projection transformations. The first step is to compute the point $\mathbf{q}_n$ on the near plane that is being projected to the current window pixel position (x,y), which is basically an inversion of the viewport transformation:

$$\mathbf{q}_n = \begin{pmatrix} x \cdot \frac{w_n}{w_{vp}} - \frac{w_n}{2} \\ y \cdot \frac{h_n}{h_{vp}} - \frac{h_n}{2} \\ -n \end{pmatrix},$$

where $w_{\{n,vp\}}$ and $h_{\{n,vp\}}$ denote the width and height of the near plane and viewport, respectively. Casting a ray from the origin (the eye) through this point and intersecting it with the splat's supporting plane yields the corresponding point $\mathbf{q}$ on the splat (cf. Fig. 2):

$$\mathbf{q} = \mathbf{q}_n \cdot \frac{\mathbf{c}^T\mathbf{n}}{\mathbf{q}_n^T\mathbf{n}}, \tag{6}$$

where $\mathbf{c}$ and $\mathbf{n}$ denote the splat's center and normal vector in eye coordinates. After computing $\mathbf{q}$, the parameter values (u,v) can easily be determined by Eq. 1. If the pixel is accepted, i.e. $u^2+v^2 \leq 1$, we compute its normal vector using Eq. 2 and derive a color by lighting it.

If technical datasets are to be rendered by surface splatting, sharp edges or corners can be represented by clipped splats, as first proposed by [PKKG03]. Based on the object-space parameter values (u,v), this method can easily be integrated into our framework. We represent a clip line in the splat's tangent space by three scalars (a,b,c), such that the point $\mathbf{q}$ is to be clipped if $au+bv+c<0$, i.e. at the cost of one dot product. Notice that this is much easier compared to [ZRB*04], since their approach requires the projection of two points representing the clip line to image space in order to do the half-space test there.

What remains to be done is correcting the pixel's depth value, that would otherwise equal the depth value of the center point all over the splat, causing blending artifacts as was first pointed out by [BK03]. Since we know the exact depth in eye-coordinates $\mathbf{q}_z$, we can derive the fragment's depth value z_{vp} by

$$z_{vp} = \frac{1}{\mathbf{q}_z} \cdot \frac{fn}{f-n} + \frac{f}{f-n}.$$

Note that this actually results in per-pixel correct depth values, in contrast to previous approaches using affine approximations to the projective mapping [BK03, ZRB*04].

4.1. Implementation Details

In order to render an elliptical Phong splat we send one (colored) vertex located at the center position $\mathbf{c}$ through the OpenGL rendering pipeline and pass the (scaled) tangential vectors ($\mathbf{u}$, $\mathbf{v}$) and the normal field ($\bar{\mathbf{n}}$, α, β) as texture coordinates. Hence, in comparison to standard elliptical splatting approaches we need 17 instead of 12 floating point values per splat. However, since we need one order of magnitude fewer splat primitives to achieve the same visual quality, the overall memory consumption is lower for Phong splats.

A vertex program transforms the splat center to window coordinates and the tangents and center normal to eye-space. It also computes the size of the image-space square to be rasterized (Eq. 5) and optionally (for closed models) a conservative backface test based on the transformed normal field. In addition, the vertex program precomputes the scaled normal $\mathbf{n} \cdot (\mathbf{c}^T\mathbf{n})^{-1}$ for the later ray intersection (Eq. 6).

In the fragment program we then need only 6 instructions to compute the local parameter values. Based on them, we either discard the fragment or compute its normal vector. For efficient per-pixel lighting, we use a cube map to store precomputed lighting terms as a function of normal vectors. This precomputation derives the lighting components that are independent from the surface color, based on lighting conditions and achromatic surface reflectance. Colored or textured models can then be rendered by multiplying these intensity values by the color of the splat or pixel, respectively. In the case of achromatic light sources (r=g=b), we can even store separate diffuse and specular intensity values in one luminance-alpha cube map.

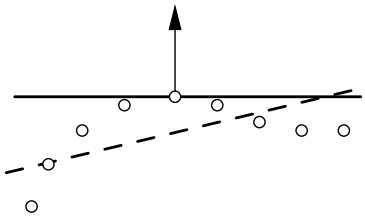

Figure 3: *A least squares plane (dashed) generally provides a better approximation to a set of sample points than a tangent plane (solid).*

When rendering models containing clipped splats, we order them such that first all non-clipped splats are rendered and then all splats with one or two clipping lines. This minimizes the overhead caused by additional splat attributes by switching the respective shader programs.

For high-quality filtering and blending we exactly follow the approach of [BK03]. We assign a radially decreasing weight to splat pixels and sum up their weighted contributions by alpha blending. In order to blend only overlapping pixels with slightly different depths, we have to use two rendering passes, the first one doing so-called visibility splatting. A final normalization, i.e. a per-pixel division by the sum of weights, is done in a pixel shader by rendering a viewport-sized quad that is textured by the outcome of the two rendering passes.

5. Discussion

Using varying normal fields for splat rendering was first proposed by [KV01]. They generate point datasets by sampling continuous smooth NURBS surfaces or triangle meshes [KV03], using a local Taylor expansion to estimate differential properties at the splat center. The geometry is approximated by the tangent plane at the splat center and normal vectors are constructed based on the principal curvatures at the center point.

In contrast, the input data we consider are just sample points with associated normal vectors, i.e. we do not require explicit surface geometry or topology information. In this sense, our approach more closely follows the spirit of point-based graphics. In addition, our splat generation phase does not even require the sample normals to be consistently oriented, as the orientation is canceled out by the homogeneous normal representation. To get consistently oriented normal fields for the rendering process, only the splats S_j (defining the offset tangent planes) should be consistently oriented, but their number is significantly smaller than the number of the input samples.

Comparing the two approximation methods, least squares planes can approximate the local geometry better than a tangent plane (cf. Fig. 3). In strictly convex/concave configurations, the approximation error of the least squares solution is roughly half of the error of the tangent plane. In the same sense our least squares linear normal field approximates better than the normal field of a quadratic surface approximation used in [KV01].

From a signal processing point of view their approach is a subsampling process, while we approximate both geometry and normals by least squares methods, which can be considered as a low-pass filtering process. Hence, our approximation effectively avoids alias artifacts, that would otherwise result in shading discontinuities.

The normal approximation of [KV01] can be considered to be the normal field of a quadratic patch parameterized over the tangent plane $f(u,v) = au^2 + bv^2 + cuv + du + ev + f$. Since at the splat center $(0,0)$ they interpolate position and normal vector, i.e. $f(0,0) = f_u(0,0) = f_v(0,0) = 0$, and since the axes $\mathbf{u}$ and $\mathbf{v}$ are aligned to principal curvature directions, i.e. $f_{uv}(0,0) = 0$, the quadratic patch actually has the form $f(u,v) = au^2 + bv^2$. In contrast, we use a linear function $n(u,v) = au + bv + c$ for the normal fitting, which corresponds to using the normal field of a general quadratic function $f(u,v) = au^2 + bv^2 + du + ev + f$. Again, the term cuv vanishes since the tangent coordinate system is aligned to the principal axes. The additional degrees of freedom d and e correspond to the adjustment of the center normal $\bar{\mathbf{n}}$ in order to improve the normal fitting. The offset f represents the fact that our least squares plane is not fixed at the splat center $\mathbf{p}_i$.

More important from a practical point of view is that our decoupling of the normal fitting from the geometry fitting leads to the simple solution of two 2×2 linear systems (Eq. 3). Our method is also more robust in the presence of imperfect real-world datasets, since the least squares method smoothly distributes the error over the whole splat, while the estimated curvature information at the splat center will get unreliable. The slight shading discontinuities caused by inevitable approximation errors of the linear normal field require the use of high-quality filtering and blending — this was not necessary for the clean datasets used in [KV01].

An important difference to other GPU based surface splatting methods is that our approach is projectively correct since it does not approximate the projection by an affine mapping. The resulting exact parameter values are important for splat shape, normal computation, and splat clipping, and they enable a simpler implementation. Using exact depth values for each pixel leads to more robust blending and avoids artifacts even for large splats which are almost perpendicular to the image plane.

On the other hand, not using an affine approximation to the projective mapping disables us to use EWA splatting, i.e. to combine object-space reconstruction filter and image-space band-limiting filter into one Gaussian. However, using the object-space reconstruction filter alone turned out to be sufficient in practice, as also mentioned in [BK03].

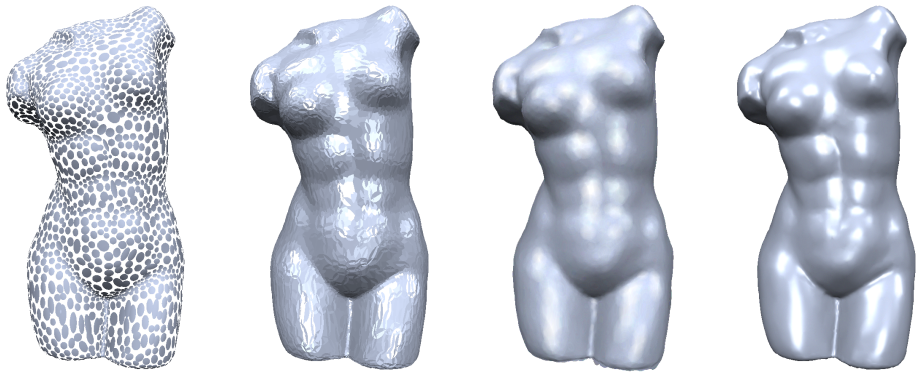

Figure 4: *The same torso dataset containing 3k splats (left) rendered using flat shading without blending (center left), flat shading + Gaussian blending (center right) and Phong splatting (right).*

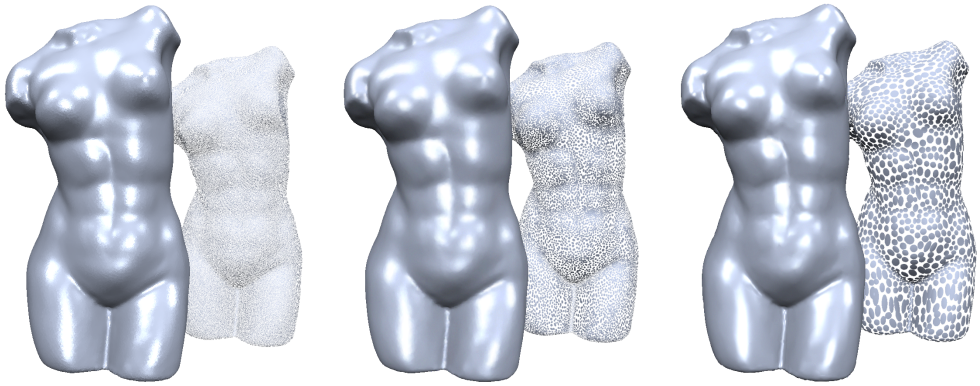

Figure 5: *To achieve comparable visual quality the model complexity has been adjusted for each rendering mode. From left to right: flat shading (170k splats), flat shading + Gaussian blending (33k splats), Phong splatting (3k splats).*

6. Results

We compare our Phong splatting system to related work both in terms of rendering quality and rendering efficiency. For the models we present, a Phong splat representation has been generated using the algorithm described in Sec. 3.

Fig. 4 shows a torso model of 3k splats using different rendering techniques. Using standard surface splatting without any filtering clearly shows the flat shading discontinuities. Gaussian blending (using two rendering passes) manages to decrease these artifacts, but also blurs the image noticeably. Phong splatting uses per-pixel lighting and clearly achieves the highest rendering quality. Note that these three results are comparable to flat shading, Gouraud shading (smooth shading but no sharp highlights) and Phong shading in the triangle mesh rendering case.

The same model is shown in Fig. 5, but the number of points has been adjusted such that each rendering mode achieves about the same visual quality. Although using a very high splat count (170k splats), the flat shading artifacts can hardly be removed without blending. Even with blending enabled, the sampling has to be quite dense (33k splats) in order to result in high visual quality and sharp highlights. Using Phong splatting, we need one order of magnitude fewer splat primitives (3k splats) in order to achieve the same visual quality. This is confirmed by Fig. 8, that shows the decrease in visual quality for several simplifications of a scanned statue. It is clearly noticeable that the rendering quality is not as tightly related to geometric complexity for Phong splatting as for standard splatting. An example of Phong shading for textured models is shown in Fig. 6.

In order to avoid shading discontinuities, [ZPvBG01] proposed to splat normal vectors in addition to colors and to compute the final pixel colors by a deferred shading of these averaged per-pixel values. However, when blending the normal vectors, the gradient of the normal field depends on the distance of neighboring splats while in the case of Phong splats we derive the normal gradient directly from the highly detailed input data (cf. Fig. 7). Additionally, this method is not suitable for an implementation on current GPUs, since

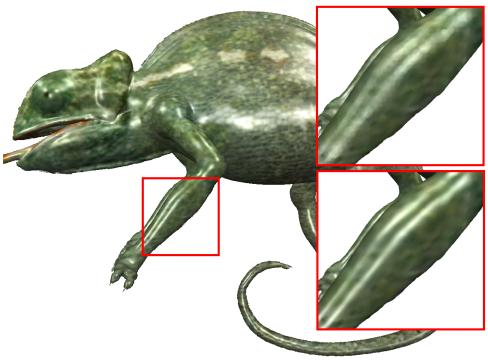

Figure 6: *A comparison of standard splatting (top closeup) to Phong splatting (bottom closeup) for the colored chameleon model consisting of 100k splats.*

the accumulated (alpha blended) normals suffer from discretization artifacts.

Finally, the combination of splat clipping and Phong splatting is shown in Fig. 9, where the well-known fandisk dataset is rendered as point-based representation using the different shading modes.

Besides considerably higher visual quality, our Phong shading system also provides splat rates comparable to recent approaches. On a 3GHz Pentium4, GeForceFX 5950, we achieve splat rates between 3.5M and 4M splats/sec for a window size of 512×512. Taking into account that we need significantly fewer splats for the same visual quality, our method can even be considered to be more efficient than previous approaches.

7. Conclusion

In this paper we proposed to use a linearly varying normal field for per-pixel lighting of surface splats. The resulting Phong splats have been shown to result in a much higher rendering quality compared to flat shaded splats. Although non-constant normal fields have been used before by [KV01], our Phong splat generation provides an easier formulation and also offers a more robust fitting of both the geometry and the normal field, which is especially important for noisy real-world datasets.

The presented Phong splatting system is implemented completely on the GPU, leading to a rendering speed of up to 4M splats/sec. Additionally, our approach results in a projectively correct rasterization with pixel-exact depth values, leading to more precise results. The example implementations of the vertex and pixels shader programs included in the supplementary material will also be available at `http://www.rwth-graphics.de`.

As general geometry processing algorithms also benefit from the lower splat counts, we believe that Phong splats can evolve to an interesting alternative to standard "flat" splats.

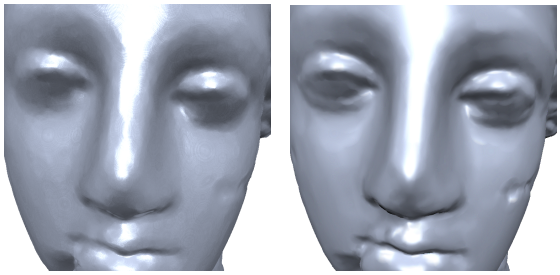

Figure 7: *A coarse model (9k splats) rendered by deferred per-pixel shading of averaged splat normals (left) and Phong splatting (right). Since the normals of the former method are not based on the detailed input data ($\mathbf{p}_i$, $\mathbf{n}_i$), highlights are not represented sufficiently. The highest frequency perturbations in the left image are due to GPU discretizations and not related to the approach in general.*

Figure 8: *The decrease in visual quality related to model complexities of 350k, 110k and 35k splats, shown as close-ups for standard splatting (left) and Phong splatting (right).*

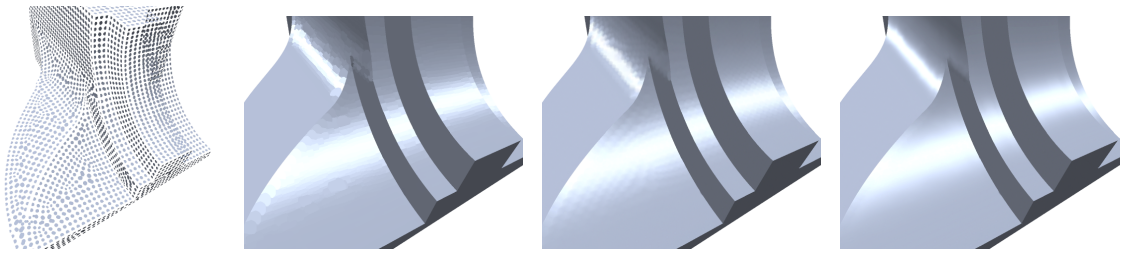

Figure 9: *Phong splatting can easily be combined with splat clipping, resulting in sharp features. A splat representation of the fandisk dataset (left) is rendered using flat shading (center left), flat shading + Gaussian blending (center right), and Phong splatting (right).*

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., SILVA C.: Point set surfaces. In *Proc. of IEEE Visualization 01* (2001), pp. 21–28. 2

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern GPUs. In *Proc. of Pacific Graphics 03* (2003). 2, 4, 5

[Bui75] BUI-TONG PHONG: Illumination for Computer Generated Pictures. *CACM 18(6)* (1975), 311–317. 1

[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *Proc. of Eurographics Workshop on Rendering 02* (2002). 2

[CH02] COCONU L., HEGE H.-C.: Hardware-accelerated point-based rendering of complex scenes. In *Proc. of Eurographics Workshop on Rendering 02* (2002), pp. 41–51. 2

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. In *Proc. of Siggraph 03* (2003). 2

[FCOAS03] FLEISHMAN S., COHEN-OR D., ALEXA M., SILVA C. T.: Progressive point set surfaces. *ACM Transactions on Graphics 22*, 4 (2003). 2

[Gar99] GARLAND M.: *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, CS Dept., 1999. 3

[GD98] GROSSMAN J. P., DALLY W. J.: Point sample rendering. In *Proc. of Eurographics Workshop on Rendering 98* (1998), pp. 181–192. 2

[GL89] GOLUB G. H., LOAN C. F. V.: *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989. 3

[KV01] KALAIAH A., VARSHNEY A.: Differential point rendering. In *Rendering Techniques 2001* (2001). 1, 2, 5, 7

[KV03] KALAIAH A., VARSHNEY A.: Modeling and rendering points with local geometry. *IEEE Transactions on Visualization and Computer Graphics 9(1)* (2003), 30–42. 2, 5

[LW85] LEVOY M., WHITTED T.: *The use of points as display primitives*. Tech. rep., CS Departement, University of North Carolina at Chapel Hill, January 1985. 2

[Paj03] PAJAROLA R.: Efficient level-of-details for point based rendering. In *Proc. of IASTED Comuter Graphics and Imaging* (2003). 2

[PG01] PAULY M., GROSS M.: Spectral Processing of Point-Sampled Geometry. In *Proc. of Siggraph 01* (2001). 2

[PGK02] PAULY M., GROSS M., KOBBELT L.: Efficient simplification of point-sampled surfaces. In *Proc. of IEEE Visualization 02* (2002). 2

[PKG03] PAULY M., KEISER R., GROSS M.: Multi-scale feature extraction on point-sampled surfaces. In *Proc. of Eurographics 03* (2003). 2

[PKKG03] PAULY M., KEISER R., KOBBELT L., GROSS M.: Shape Modeling with Point-Sampled Geometry. In *Proc. of Siggraph 03* (2003). 2, 4

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proc. of Siggraph 00* (2000), pp. 335–342. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: a multiresolution point rendering system for large meshes. In *Proc. of Siggraph 00* (2000), pp. 343–352. 2

[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. 2

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proc. of Eurographics Workshop on Rendering 01* (2001), pp. 151–162. 2

[WK04] WU J., KOBBELT L.: Optimized subsampling of point sets for surface splatting. In *Proc. of Eurographics 04* (2004). 2, 3

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: PointShop 3D: An Interactive System for Point-Based Surface Editing. In *Proc. of Siggraph 02* (2002). 2

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. of Siggraph 01* (2001), pp. 371–378. 2, 6

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Proc. of Graphics Interface 04* (2004). 2, 4