Normal Mapping for Surfel-Based Rendering

Mathias Holst
University of Rostock
Albert-Einstein-Str. 21
18059 Rostock, Germany
mholst@informatik.uni-rostock.de

Heidrun Schumann
University of Rostock
Albert-Einstein-Str. 21
18059 Rostock, Germany
schumann@informatik.uni-rostock.de

ABSTRACT

On the one hand normal mapping is a common technique to improve normal interpolation of low tesselated triangle meshes for a realistic lighting. On the other hand today's graphics hardware allows texturing of view plane aligned point primitives. In this paper we illustrate how to use textured points together with normal mapping to increase surfel splatting quality, especially when using larger splats on lower level of detail. In combination with a silhouette refinement this results in a significant decimation of needed surfels with small visual disadvantages only. Furthermore, we explain how to create a normal map for points within a point hierarchy.

Keywords

Normal Mapping, Surfel Splatting, Point-Based Rendering, GPU-Programming.

1 INTRODUCTION

In recent years point-based rendering has been proven to be effective and efficient for rendering highly detailed complex geometric models. Point-based rendering bases on the idea, that polygonal representations get less efficient with increasing polygon number, because in this case each polygon covers only a few pixels in image space [LW85]. Additionally triangle meshes, or polygonal meshes in general, are not easy to handle and to simplify because of their connectivity. Points on the other side do not have any connectivity and can be stored and merged very easily using simple subdivision schemes [PGK02].

Since points only have a position but no dimension, they are parameterized with other attributes that describes their look. Usually these are a normal and a radius to represent circular disks in 3D (see fig. 1), known as *surfels* (from surface elements). With surfels a dense, opaque and smooth surface approximation can be described.

To get a high quality rendering result small and many surfels have to be used. However, the number of vertices (e.g. points) that is processed by the GPU is a framerate limiting factor. Thus, it is useful to render fewer but larger surfels instead. To attenuate the loss of surface features we propose to apply normal mapping. This is possible, because today's graphics hardware allows texturing point primitives. This texturing is given for image space only. Therefore we show in this paper how object-space texture coordinates can be obtained by adding only a 2D texture coordinate and a scaling factor as additional surfel attributes. Of course, normal mapping does not decimate the total number of pixels/fragments to shade but in the case of non-parallel vertex-fragment processing this results in a significant framerate increase.

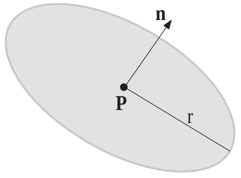

Figure 1: Surfel geometry.

When working with normal mapping normal maps have to be associated to the lower levels of detail of the original object. In this paper an algorithm is presented to get normal maps for surfels of point hierarchies. In past several point hierarchies has been developed. Therefore, we take the two most popular hierarchies into account: point and hybrid bounding-sphere hierarchies.

Since normal mapping only affects the shading of pixels, but not the shape of the underlying geometry, the silhouette looks very coarse when using low tesselated polygonal or point-based models. Therefore, it is useful to enhance the rendering process by silhouette refinement using more primitives at these surface parts. In contrast to polygonal-based approaches silhouette refinement can be easily done using point hier-

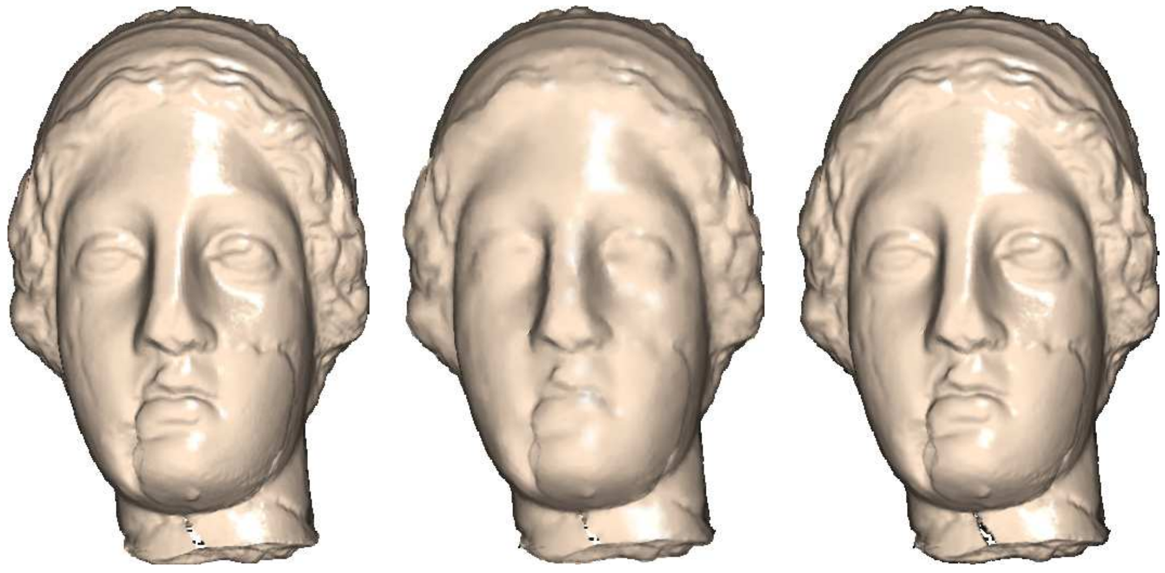

Figure 2: Rendering of the Igea model with 255k surfels (a). Same model rendered with only 32k surfels and silhouette refinement (b). Same LOD rendered using normal mapping (c).

archies, because no connectivity has to be considered. We use normal cones for this purpose, which results in a high-quality rendering as it can be seen in fig. 2(c).

2 Previous Works

Relevant works that inspired this paper can be categorized into three groups: surfel-based rendering, point hierarchies and point-based silhouette refinement.

Surfel-Based Rendering In general a surfel is an oriented (n-1)-dimensional oriented object in n-dimensional space [Her92]. Using surfels as rendering primitives was first proposed by [PZvBG00]. Here, objects are rendered by a two pass approach: First all visible surfels have to be estimated using z-buffer and secondly, these surfels have to be rendered, called splatting. This splatting was improved in [ZPvBG01] by applying a gaussian elliptical kernel as alpha mask for each surfel in screen space to meet the Nyquist criterion. In [RPZ02] an hardware supported version is proposed.

We use a similar technique as proposed in [BK03], which includes a complete hardware support for filtering and splatting with a minimum CPU overhead. Moreover we decrease the number of pixel that have to be shaded by using a more sophisticated splat size measure in screen space.

Point Hierarchies Continuous LOD is nearly always integrated in point-based rendering approaches. This yields from the computational simplicity to create point hierarchies using subdivision schemes. A comparison of point cloud simplification can be found in [PGK02].

Since point clouds are often generated by sampling triangle meshes, these meshes can be used as alternative surface representation on the highest LOD. Such hybrid hierarchies are proposed in [CN01] and [DVS03] for example. In [CAZ01] a triangle-based multi-resolution hierarchy is used for higher LOD and a point hierarchy on top for lower LOD.

In this paper we propose algorithms to generate normal maps for both kinds of point hierarchies, but in our implementation we only use a pure point hierarchy created from an octree-based space subdivision.

Silhouette Refinement In most point-based frameworks the whole surface is approximated by nearly equally sized points/surfels to guarantee feature preservation also in the interior without a special silhouette refinement. A first framework that realizes silhouette refinement is the POP system [CN01]. In this hybrid system triangles are used as highest available LOD. If the normal of a surfel is nearly perpendicular to the viewing vector, its triangle children will be rendered. This seems to be too conservative, especially if the object is very small in screen space. In this case triangles cover a few pixels only. Therefore, this may results in aliasing artifacts.

Another system that uses silhouette refined surface approximations was proposed in [LH01]. Here the QSplat system [RL00] was extended to use normal cones for silhouette detection and refinement. Using a sophisticated perceptual model, surfels lying on the silhouette are assumed to cause high frequency and contrast in the final image. Thus, they are skipped and the hierarchy is further traversed top-down until the expected rendering result is indistinguishable from the original. This is an effective approach. Therefore, we also use normal cones to detect surfels lying on the silhouette. We suggest a more simple (and faster) LOD selection using two radii that limit surfel size for silhouette and interior surfels. However, without high effort this can be extended to use the perceptual model of Luebke and Hallen.

3 Surfel Rendering

In this section we discuss, how surfels are splatted to render the object surface and how we integrate normal mapping into this procedure.

3.1 Splat Sizing

Using today's graphics hardware points are rendered as view-plane aligned squares (e.g. aliased OpenGL points). Thus, for each surfel its bounding square size has to be calculated either on the CPU or on the GPU by a vertex shader. Assuming a perspective projection this size depends basically on surfel eye-space z-value z_{eye} and orientation. However, most approaches only consider z_{eye} and assume a view-plane aligned surfel. But when texturing surfels the size should be as exact as possible to prevent an unrealistic texture scaling. Therefore, also surfel orientation should be considered. Our method is similar to [ZRB$^+$04] but mathematically easier. We approximate the surfel shape in eye-space by a rotated ellipse function in 2D and calculate its bounding-box. Then the largest dimension of this box is projected to image space to get the bounding square size in screen-space. In doing so, up to 50% smaller splat area can be defined, as illustrated in fig. 3(a).

An ellipse curve with x-axis radius a and y-axis radius b positioned at the origin can be described by:

$$f(x) \;=\; b \cdot \sqrt{1-\frac{x^2}{a^2}} \tag{1}$$

To get the bounding box of an ellipse which is rotated by angle γ we have to calculate the maximum y and x values. One way to get these values is to calculate the points on f with derivation $f'(x_1) = -\tan(\gamma)$ and $f'(x_2) = \cot(\gamma)$ and rotate them back by γ. These points on f are defined by $(x_1, f(x_1))^T$ and $(x_2, -f(x_2))^T$, with

$$\begin{aligned} x_1 &= \frac{-a^2\tan(\gamma)}{\sqrt{a^2\tan(\gamma)^2+b^2}} \\ x_2 &= \frac{a^2\cot(\gamma)}{\sqrt{a^2\cot(\gamma)^2+b^2}}. \end{aligned} \tag{2}$$

After rotating these points by γ we get the maximum x and y-value by:

$$\begin{aligned} x_{max} &= \cos(\gamma)x_2+\sin(\gamma)f(x_2) \\ y_{max} &= \sin(\gamma)x_1+\cos(\gamma)f(x_1) \end{aligned} \tag{3}$$

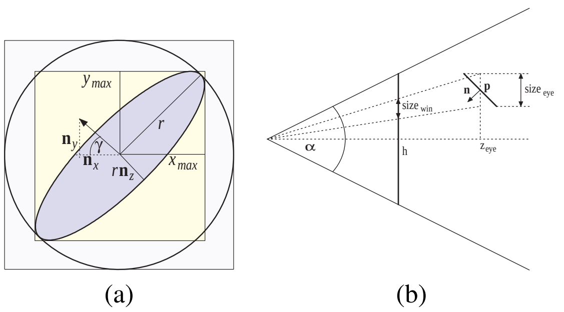

Figure 3: Comparison of bounding squares of surfel's bounding sphere vs. ellipse approximation (a). Projection of eye space square length to screen space (b).

.

For an oriented surfel in 3D that is transformed to eye-space with eye-space orientation $\mathbf{n}$ and radius r you get $a = r$, $b = r\mathbf{n}_z$ and γ is given by the adjacent leg $\mathbf{n}_x$ and opposite leg $\mathbf{n}_y$ (fig. 3(a)). This yields to

$$x_{max} = r\sqrt{1-\mathbf{n}_x^2}, \quad y_{max} = r\sqrt{1-\mathbf{n}_y^2} \tag{4}$$

Using this we finally get the point-square dimension in eye-space by:

$$size_{eye} = 2max(x_{max}, y_{max}) = 2r\sqrt{1-\min(\mathbf{n}_x,\mathbf{n}_y)^2}. \tag{5}$$

After approximating the eye-space square size, this size is projected into image-space. Assuming that view frustum's aspect ratio equals viewport's aspect ratio this is done by

$$size_{win} = \frac{size_{eye}}{z_{eye}} \cdot \frac{h}{2\tan(\frac{\alpha}{2})}, \tag{6}$$

where h is the viewport height in pixel and α the field-of-view angle as illustrated in fig. 3. We totally compute $size_{win}$ by a vertex shader, which only needs a few instructions more than applying $size_{eye} = 2r$.

3.2 Splat Shaping and Filtering

After resizing the surfel, it is rendered as a view-plane aligned square. We use the rendering scheme proposed in [BK03] to get a high quality anti-aliased rendering result without the typical thickening effect of square splats. In the following this procedure is briefly described.

Today's graphics cards are able to compute a texture coordinate $\mathbf{t} \in [-1,1]^2$ for each splat pixel. Together with the eye-space surfel normal $\mathbf{n}$ a depth offset $\mathbf{t}_z$ for every pixel can be computed:

$$\mathbf{t}_z \;=\; -\frac{\mathbf{n}_x}{\mathbf{n}_z}\mathbf{t}_x - \frac{\mathbf{n}_y}{\mathbf{n}_z}\mathbf{t}_y \tag{7}$$

as illustrated in fig. 4(a). If $||\mathbf{t}||$ is less than one the pixel belongs to the eye-space ellipse area. By using a

Gaussian kernel $\mathscr{G}$ for every surfel an alpha value for this pixel can be computed by $\mathscr{G}(||\mathbf{t}||)$. This forms an ellipse with a smooth alpha value falloff at the border.

If several splats overlap in image-space they are blended, but only if their z-value in eye-space is sufficiently small. In this case they define a contiguous surface part. Otherwise splats in front should overdraw splats behind. Blended pixel values are summed up weighted with their alpha values (also known as fuzzy splatting). In the ideal case, these weights sum up to one, forming an opaque surface. Because this is not the general case additionally a per pixel normalization is needed. Efficient algorithms for this purpose using the possibilities of modern graphics hardware are described in [BK03].

3.3 Normal Mapping for Surfels

Our goal is to increase surfel splatting quality by using normal maps for pixel shading. Therefore, we need texture coordinates for every pixel of the surfel square (resp. on the surfel disk). For polygonal meshes texture coordinates are given for every vertex, and after rasterization for every pixel its texture coordinate is interpolated. When texturing surfel splats, for every pixel such an interpolation is not possible, because surfels are only described by one parameterized vertex. Thus, we calculate this texture coordinate using the given pixel parametrization $\mathbf{t}$ (see last section 3.2) particulary calculated by the graphics card together with a texture coordinate $\mathbf{t}_s$ and a scaling factor w_s for the normal map which are static for the whole surfel and which are passed as vertex/fragment attribute. How these values can be determined is explained in section 4.3.

Since for every surfel only its normal $\mathbf{n}$ is given to describe its orientation, no exact mapping is possible from $\mathbf{t}$ to the surfel plane. Instead there is a circle of possible solutions. Thus, we need another orientation normal $\mathbf{o}_1$, which is orthogonal to $\mathbf{n}$ and describes the rotation angle around $\mathbf{n}$. We suggest to compute $\mathbf{o}_1$ by a simple scheme:

$$\mathbf{o}_1 = \begin{cases} \left(\frac{\mathbf{n}_z}{\sqrt{1-\mathbf{n}_y^2}}, 0, \frac{-\mathbf{n}_x}{\sqrt{1-\mathbf{n}_y^2}}\right)^T & ,\text{if } |\mathbf{n}_y| < 1 \\ (0,0,1)^T & ,\text{else.} \end{cases} \tag{8}$$

In addition a third orthonormal vector $\mathbf{o}_2$ can be calculated by

$$\mathbf{o}_2 = \frac{\mathbf{o}_1 \times \mathbf{n}}{||\mathbf{o}_1 \times \mathbf{n}||} \tag{9}$$

to define a local coordinate system on the surfel plane with projection matrix $\mathbf{S} = [\mathbf{o}_1\ \mathbf{o}_2\ \mathbf{n}]$, as illustrated in fig. 5. Note, that $\mathbf{o}_1$ and $\mathbf{o}_2$ are generic and can also be computed in a vertex shader from object-space surfel normal $\mathbf{n}$ without additional attributes and memory effort.

When projecting a surfel to eye-space using modelview matrix $\mathbf{M}$ this local coordinate system is projected to eye-space, too, by $\mathbf{S}' = \mathbf{SM}$. To get the texture coordinate in surfel space, $\mathbf{t}$ has to be projected back by $\mathbf{t}' = \mathbf{S}'^{-1}\mathbf{t}$ as it can be seen in fig. 4(b). Since $\mathbf{t}'$ lies on the surfel plane $\mathbf{t}'_z = 0$. Hence, only the first two rows of $\mathbf{S}'^{-1}$ have to be computed, which can be done by:

$$\frac{1}{|\mathbf{S}'|}\begin{bmatrix} \begin{vmatrix} \mathbf{o}_{2_y} & \mathbf{n}_y \\ \mathbf{o}_{2_z} & \mathbf{n}_z \end{vmatrix} & \begin{vmatrix} \mathbf{n}_x & \mathbf{o}_{2_x} \\ \mathbf{n}_z & \mathbf{o}_{2_z} \end{vmatrix} & \begin{vmatrix} \mathbf{o}_{2_x} & \mathbf{n}_x \\ \mathbf{o}_{2_y} & \mathbf{n}_y \end{vmatrix} \\ \begin{vmatrix} \mathbf{n}_y & \mathbf{o}_{1_y} \\ \mathbf{n}_z & \mathbf{o}_{1_z} \end{vmatrix} & \begin{vmatrix} \mathbf{o}_{1_x} & \mathbf{n}_x \\ \mathbf{o}_{1_z} & \mathbf{n}_z \end{vmatrix} & \begin{vmatrix} \mathbf{n}_x & \mathbf{o}_{1_x} \\ \mathbf{n}_y & \mathbf{o}_{1_y} \end{vmatrix} \end{bmatrix}. \tag{10}$$

If $||\mathbf{t}|| < 1$ then surfel base texture coordinate $\mathbf{t}'$ is in $[-1,1]^2$. Thus, $(\mathbf{t}'_x, \mathbf{t}'_y)^T$ can be used easily to get the final normal map texture coordinate $\mathbf{t}_n$ of the shaded pixel using a linear mapping:

$$\mathbf{t}_n = \frac{w_s - 1}{2}\mathbf{t}' + \mathbf{t}_s, \tag{11}$$

where w_s is the width (resp. height) of the area a surfel disk covers in the normal map and $\mathbf{t}_s$ is the surfel texture coordinate in the center of this area (fig. 4(c)).

Finally, $\mathbf{t}_n$ can be used to address the texel in the normal map, that contains the normal to use for shading the surfel at this pixel, as illustrated in fig. 4(d).

4 Normal Map Estimation

After developing a rendering algorithm for surfel splats using normal mapping now we explain how a normal map for different surfel hierarchies can be generated.

Recent point-based level of detail approaches either use a large point set [RL00] or a triangular mesh (e.g. [CN01]) to describe the object on the highest LOD. Based on this model a point tree is generated by a subdivision scheme. In the next sections we give algorithms for normal map creation for point and hybrid point hierarchies.

4.1 Normal Map for Point Hierarchies

To create a normal map of a point hierarchy two steps have to be performed for every surfel: Firstly, the surfel has to be rasterized and secondly for every raster point a ray has to be shot to obtain all surfels on the highest LOD (called base surfels, $\mathscr{S}_{base}$) that affect the normal of the raster point.

For rasterization of a surfel s the same orientation vectors $\mathbf{o}_1$, $\mathbf{o}_2$ are applied as for normal mapping (see section 3.3). If an area of $w_s \times h_s$ texel is preserved in the normal texture for surfel s for every texel $(x,y) \in [0,w_s) \times [0,h_s)$ a position on the surfel disk is given by:

$$\mathbf{P}_{x,y} = \mathbf{P}_s + r_s\left(\frac{2x+1}{w_s} - 1\right)\mathbf{o}_1 + r_s\left(\frac{2y+1}{h_s} - 1\right)\mathbf{o}_2 \tag{12}$$

where r_s is the surfel radius, as illustrated in fig. 6.

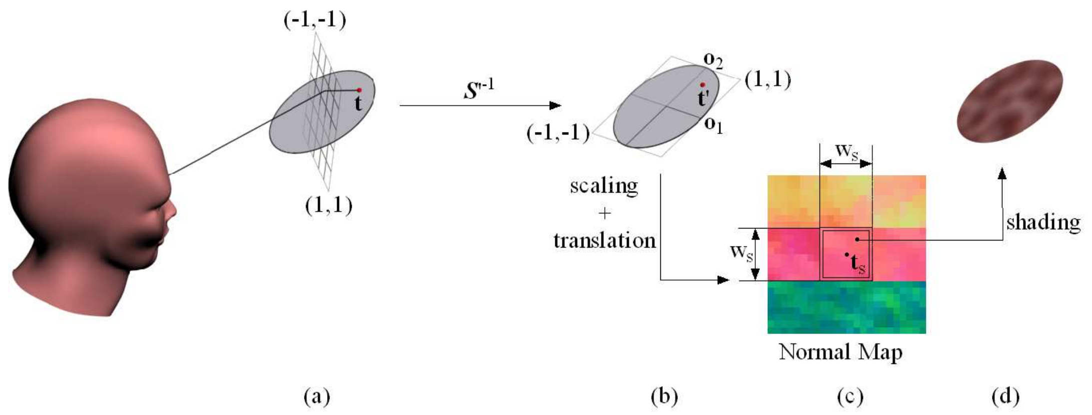

Figure 4: Normal mapping steps: Original texture coordinate t with calculated depth value (a). Texture coordinate t′ projected to surfel space (b). Mapping to normal map texture coordinate t_n (c). Final result of surfel shading (d).

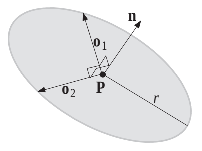

Figure 5: Extended surfel geometry.

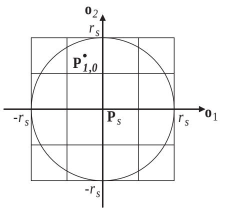

Figure 6: Rasterization of a surfel disk using a uniform raster.

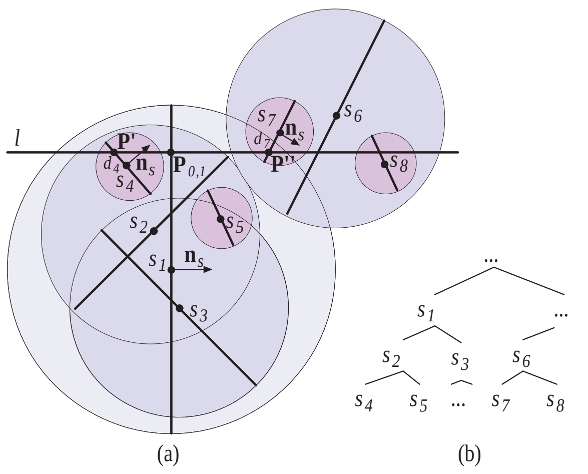

Figure 7: Geometry for line-surfel intersection (here in 2D). Line l intersects base surfels s_4,s_7 and s_8 but only s_4,s_7 are used for normal estimation at point $\mathbf{P}_{0,1}$ due to additional bounding sphere test (a). The corresponding point hierarchy (b).

To estimate the normal for every texel we choose a simple raycasting approach. We estimate all surfels on the highest LOD, which intersect the line l given by position $\mathbf{P}_{x,y}$ and surfel normal $\mathbf{n}_s$. Note that we do not use a ray, because base surfels "behind" s have to be considered, too. This is illustrated in fig. 7(a) in 2D for a raster position $\mathbf{P}_{0,1}$ of surfel s_1 for which line l intersects base surfels s_4,s_7 and s_8.

Base surfels that are far away from $\mathbf{P}_{x,y}$ should not be considered for the normal at this point, because they do not belong to the surface part approximated by s. A first attempt could only consider base surfels that are also children of s (in fig. 7(a) this is only s_4). But this is not sufficient generally, because surfels overlap. Thus, we choose a top down approach. Starting at the root surfel of the point hierarchy, surfels are determined top down, whose bounding sphere intersects with the bounding sphere of s (i.e. s_1 in fig. 7(a)). If a surfel is also a base surfel and it intersects line l it will be considered for normal computation at point $\mathbf{P}_{x,y}$. In fig. 7(a) these are s_4 and s_7. This algorithm is very fast, because for ray intersection the space subdivision given by the point hierarchy is used (fig. 7(b)). Thus, only a small number of surfels have to be tested for bounding sphere intersection and line intersection, respectively.

After a set of base surfels $\mathscr{S}_{x,y} \subseteq \mathscr{S}_{base}$ is found for a given pixel $\mathbf{P}_{x,y}$ on surfel s, the normal $\mathbf{n}_{x,y}$ at position $\mathbf{P}_{x,y}$ is estimated from this. Widely used for this purpose is a weighted and normalized sum:

$$\mathbf{n}_{x,y} = \frac{\sum_{s' \in \mathscr{S}_{x,y}} w_{s'} \mathbf{n}_{s'}}{\sum_{s' \in \mathscr{S}_{x,y}} w_{s'}}, \tag{13}$$

where $w_{s'}$ weights the contribution of each base surfel. Since splats are blended in image space using an alpha mask defined by a Gaussian $\mathscr{G}_s$ for every surfel s (see section 3.2), we choose this Gaussian to get the weights $w_{s'}$. Therefore the distance $d_{s'}$ of the base surfel s' center to the intersection point with line l is mea-

Algorithm 1 Algorithm to get the normal for a line by intersection with base surfels.

```
SurfelLineIntersec(Line $l$, Surfel $s_1$, Surfel $s_2$, Normal $\mathbf{n}$)
  $d := ||\mathbf{P}_{s_1} - \mathbf{P}_{s_2}||$;
  if ($d < r_{s_1} + r_{s_2}$)
      // bounding spheres of $s_1$ and $s_2$ intersect
      if ($s_2 \in \mathscr{S}_{base}$)
          $\mathbf{P}$ := IntersectionPoint($l$, $s_2$);
          $d := ||\mathbf{P} - \mathbf{P}_{s_2}||$;
          if ($d < r_{s_2}$)
              // $l$ intersects $s_2$
              $\mathbf{n} := \mathbf{n} + \mathscr{G}_{s_2}(d) \cdot \mathbf{n}_{s_2}$;
          end if
      else
          for each child surfel $c$ of $s_2$ do
              SurfelLineIntersec($l$, $s_1$, $c$, $\mathbf{n}$);
          end for
      end if
  end if
end

NormalForLine(Line $l$, Surfel $s$, Normal $\mathbf{n}$)
  $\mathbf{n} := (0,0,0)^T$;
  SurfelLineIntersec($l$, $s$, $rootSurfel$, $\mathbf{n}$);
  $\mathbf{n} := \mathbf{n}/||\mathbf{n}||$;
end
```

sured (see d_4 and d_7 in fig. 7(a)). Then the weights $w_{s'}$ are given by:

$$w_{s'} = \mathscr{G}_s(d_{s'}). \tag{14}$$

The final procedure to get the normal $\mathbf{n}_{x,y}$ of a raster point $\mathbf{P}_{x,y}$ that forms together with the surfel normal $\mathbf{n}$ a line is summarized in algorithm 1. Finally the normal $\mathbf{n}_{x,y}$ is coded to RGB values and stored in the normal map.

4.2 Normal Map for Hybrid Hierarchies

If the highest available LOD in the hierarchy is a triangular mesh, then an algorithm similar to that for pure point hierarchies will be used. Since triangles do not overlap in well-formed triangular meshes only the triangle that intersects the line l through raster point $\mathbf{P}_{x,y}$ have to be found, instead of a set of base surfels. If this triangle $t = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ was found, the normal at the intersection point is interpolated using barycentric coordinates $\mathbf{c} = (u, v, w)^T$ at this point:

$$\mathbf{n}_{x,y} = u\mathbf{n}_\mathbf{A} + v\mathbf{n}_\mathbf{B} + w\mathbf{n}_\mathbf{C} \tag{15}$$

The pseudo-code for this procedure is shown in algorithm 2.

4.3 Normal Map Size

To create a normal map for a point hierarchies a raster size (w_s, h_s) has to be selected for every surfel s in addition to the algorithms described before. Since surfel disks are circular it is natural to choose $h_s = w_s$. For base surfels of a pure point hierarchy we only need one pixel ($w_s = 1$), to store the surfel normal itself. For every inner surfel of the point hierarchy the same w_s can be choosen, because surfel size in image space is limited by a quality threshold (see section 5). As it can be seen in fig. 8 this w_s should exceed at least the half of this threshold to preserve features.

Algorithm 2 Algorithm to get the normal for a line by intersection with a triangle that is a surfel child node.

```
PrimitiveIntersec(Line $l$, Surfel $s$, Primitive $p$, Normal $\mathbf{n}$)
  $B$ := BoundingSphere(p);
  $d := ||\mathbf{P}_s - \mathbf{M}_B||$;
  if ($d < r_s + r_B$)
      // bounding spheres of $s$ and $p$ intersect
      if ($p$ is triangle)
          if ($l$ intersects $p$)
              $\mathbf{P}$ := IntersectionPoint($l$, $p$);
              $\mathbf{C}$ := BarycentricCoordinates($\mathbf{P}$, $p$);
              $\mathbf{n} := u_\mathbf{C}\mathbf{n}_{p_A} + v_\mathbf{C}\mathbf{n}_{p_B} + w_\mathbf{C}\mathbf{n}_{p_C}$;
          end if
      else
          for each child surfel $c$ of $p$ do
              PrimitiveIntersec($l$, $s$, $c$, $\mathbf{n}$);
          end for
      end if
  end if
end

NormalForLine(Line $l$, Surfel $s$, Normal $\mathbf{n}$)
  PrimitiveIntersec($l$, $s$, $rootSurfel$, $\mathbf{n}$);
end
```

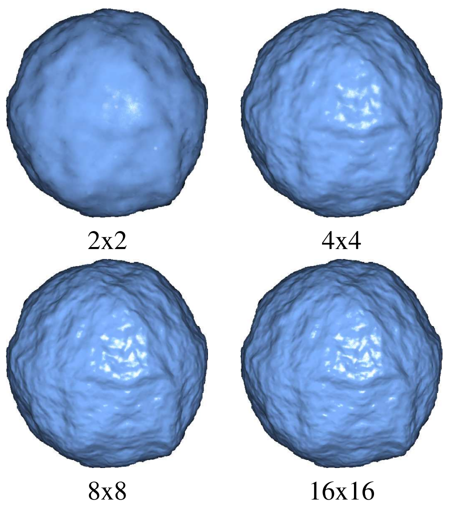

Figure 8: Comparisons of images of a meteoroid model allowing surfels up to a radius of 16 pixel in the interior and using different normal map sizes.

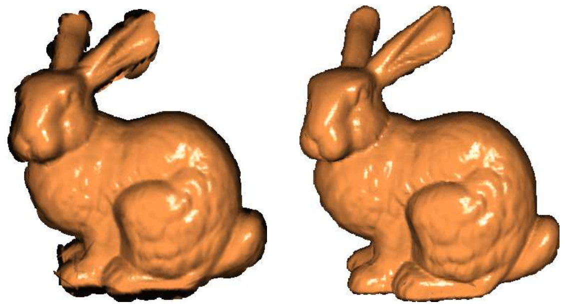

Figure 9: Bunny rendered using normal mapping without silhouette refinement (a). Same rendering with silhouette refinement (b).

Since today's graphics cards allow non power-of-two texture sizes, a proper normal map size w_{nm}, h_{nm} can be determined by:

$$w_{nm} = \left\lceil \sqrt{|\mathscr{S} \setminus \mathscr{S}_{base}|} \right\rceil w_s \tag{16}$$

$$h_{nm} = \left\lceil \frac{|\mathscr{S}_{base}|}{w_{nm}} \right\rceil + w_{nm}. \tag{17}$$

If h_{nm} exceeds the maximum supported texture size, the normal map will have to be split to multiple textures. In this case for every surfel a normal map index is needed in addition.

After finding a proper normal map size, for every surfel s, its texture coordinate t_s can be assigned, that is used for normal mapping as described in section 3.3. The surfel size in normal map (w_s, h_s) can be assigned as additional surfel attribute. But since it is static we decide to pass it as constant to the fragment shader for texturing.

5 Silhouette Refined LOD Selection

As known from multi-resolution techniques for polygonal meshes even the best texturing does not prevent a rough looking silhouette when choosing a low tesselated model. The same problem appears in point-based rendering using normal maps, as illustrated in fig. 9(a). Thus, the silhouette has to be rendered using smaller but more splats (fig. 9(b)).

Detecting the exact global silhouette is complex and computational slow, therefore we apply a local silhouette estimation using normal cones. This is very fast but as expected in some case surface parts within the object are wrongly specified to lie at the silhouette.

A normal cone is a spherical cap of the unit sphere that can be described by a normal and an opening angle. To get surfels on the silhouette every surfel s contains a normal cone as additional attribute, which contains its normal and the normal cones of all surfels below s in the hierarchy. To get this normal cone, we use the algorithm developed in [BE05]. In case of a perspective view with normalized viewing vector $\mathbf{d}$ and field-of-view angle α a normal cone $(\mathbf{n}, \beta)$ contains frontfacing normals if $\mathbf{nd} \leq \sin(\alpha + \beta)$ and backfacing normals if $\mathbf{nd} > -\sin(\alpha + \beta)$, respectively. A surfel s belongs to the silhouette if its normal cone contains frontfacing and backfacing normals. We can also very efficiently integrate backface culling by culling all surfels, whose normal cone only contains backfacing normals.

Based on this, a top down LOD selection algorithm can be constructed according to [RL00] using two maximum sizes r_{sil}, r_{inner} for surfels at and not at the silhouette (see alg. 3). Note, that base surfels never lie on the silhouette, because their normal cones only contain one normal. However, this can be ignored applying top down traversal, because base surfels are always drawn if they are reached.

Algorithm 3 Silhouette refined LOD selection using radius r_{sil} to limit size for silhouette surfels and r_{inner} for silhouette surfels, respectively.

```
TraverseHierarchy(Surfel $s$)
  if normal cone of $s$ contains front facing normals
    if ($s \in \mathscr{S}_{base}$)
      DrawSplat($s$);
    else
      $r$ := size of $s$ in viewport;
      $bf$ := n.c. of $s$ contains back facing normals;
      if $(bf \wedge r \leq r_{sil}) \vee (!bf \wedge r \leq r_{inner})$
        DrawSplat($s$);
      else
        for each child surfel $c$ of $s$ do
          TraverseHierarchy($c$);
        end for
      end if
    end if
  end if
end
```

6 Implementation and Results

We have implemented our framework on a Athlon64 system with a NVidia Geforce 6800 graphics card using OpenGL. For texturing we calculate the inverse surfel base matrix $\mathbf{S}'^{-1}$ (see equ. 10) for every surfel in the vertex shader and pass it as fragment attribute. Note, that $\mathbf{S}'^{-1}$ is generic and we only need the surfel normal for its calculation. The surfel texture coordinate $\mathbf{t}_s$ and the scaling factor w_s are coded into one additional 3D vector and passed as additional vertex attribute to the vertex/fragment shader.

In table 10 you can see an exemplary framerate comparison for the Igea model that shows the number of surfels in relation to the framerate with and without normal mapping. If using no normal mapping also the interior of the object has to be rendered by many surfels. Thus, the same threshold radius to limit surfel size as for the silhouette (alg. 3) has to be choosen. As you

w_s	size	surfels	fps
2	0	250k	20
2	0.34M	250k	17
4	1.1M	119k	37
8	4.1M	70k	59
16	16.2M	49k	85

Figure 10: Framerate results for using normal mapping vs. no normal mapping (first row) when rendering the Igea model in 1024x1024. The surfel size is limited to $r_{inner} = w_s$ (e.g. $r_{sil} = 2$). The final images for the first row is shown in 2(a) and for the last row in 2(c).

can see we benefit from less surfels, which yields to a significant increase in framerate especially when using larger normal maps.

This table also shows the normal mapping overhead caused by additional vertex/fragment shader operations used for $\mathbf{S}'^{-1}$ and texturing (first and second row). On the one hand calculating $\mathbf{S}'^{-1}$ needs many operations in the vertex shader. On the other hand the number of vertices is very decimated, which more than compensates these overhead. On the other side in the fragment stage not much more fragments have to be shaded when using fewer large splats than many small. In addition, texturing surfels only needs a few additional operations (see 3.3) in the fragment shader. This is important, because fragment shading is still a bottleneck especially of point-based rendering. We can conclude that normal mapping only causes an appreciable overhead by the required texture memory for the normal map.

7 Conclusion and Future Work

In this paper we proposed an approach to decrease the number of surfels in the interior of the object surface without visual disadvantages by using normal mapping and silhouette refinement. This results in a significant increases in framerate. We also shown how to create normal maps for several types of point hierarchies using a ray-casting approach. This is accelerated by using the recursive space subdivision provided by the point hierarchy.

Although our framework supports nearly all kind of point-based surface descriptions there are opinions for future works. When needing more than one normal map (e.g. in the case of many surfels in the hierarchy) an intelligent grouping of surfels to be rendered is necessary to avoid many graphic library procedure calls to select the proper map. Another working topic is to save texture memory. One way to achieve this is to estimate surfels with nearly the same normal map or with a normal map that can be tiled. Such surfels can mostly be found on a surface part with no or low curvature.

REFERENCES

[BE05] G. Barequet and G. Elber. Optimal bounding cones of vectors in three dimensions. *Inf. Process. Lett.*, 93(2):83–89, 2005.

[BK03] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern gpus. In *PG'03 conf.proc.*, page 335. IEEE Computer Society, 2003.

[CAZ01] J. D. Cohen, D. G. Aliaga, and W. Zhang. Hybrid simplification: combining multi-resolution polygon and point rendering. In *Vis'01 conf.proc.*, pages 37–44. IEEE Computer Society, 2001.

[CN01] B. Chen and M. X. Nguyen. Pop: a hybrid point and polygon rendering system for large data. In *Vis'01 conf.proc.*, pages 45–52. IEEE Computer Society, 2001.

[DVS03] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. *ACM Trans. Graph.*, 22(3):657–662, 2003.

[Her92] G.T. Herman. Discrete multidimensional jordan surfaces. *CVGIP: Graph. Models Image Process.*, 54(6):507–515, 1992.

[LH01] D. Luebke and B. Hallen. Perceptually driven interactive rendering. Technical Report #CS-2001-01, University of Virginia, 2001.

[LW85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, 1985.

[PGK02] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02 conf.proc.*, pages 163–170, Washington, DC, USA, 2002.

[PZvBG00] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: surface elements as rendering primitives. In *SIGGRAPH'00 conf.proc.*, pages 335–342, New York, NY, USA, 2000.

[RL00] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH'00 conf.proc.*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[RPZ02] L. Ren, H. Pfister, and M. Zwicker. Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering, 2002.

[ZPvBG01] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *SIGGRAPH'01 conf.proc.*, pages 371–378. ACM Press, 2001.

[ZRB$^+$04] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly. Perspective accurate splatting. In *GI'04 conf.proc.*, pages 247–254, University of Waterloo, 2004.