Abstract

Point set surfaces (PSS) are a smooth manifold surface approximation from a set of sample points. The surface definition is based on a projection operation that constructs local polynomial approximations and respects a minimum feature size. We present techniques for ray tracing PSSs. For the computation of ray-surface intersection the properties of the projection operation are exploited: The surface is enclosed by a union of minimum feature size spheres. A ray is intersected with the spheres first and inside the spheres with local polynomial approximations. Our results show that 2–3 projections are sufficient to accurately intersect a ray with the surface.

Keywords: ray tracing, point sampled geometry, point set surfaces

1 Introduction

Point sets without additional topological information become an increasingly popular representation of surfaces (see [11, 28]). In many cases, real world shapes are sampled using scanners resulting in an initial surface representation consisting of points [18, 23]. Points are also a suitable primitive for the display of surfaces, especially as the complexity of the representation results in triangles being rendered into less than a single pixel for the final display [21, 24, 15, 7, 6, 29].

Ray tracing is still the rendering method of choice for high quality display of reflective or transmissive surfaces. Recently, ray tracing gains popularity as it is an output sensitive rendering technique and, thus, avoids the overhead of rendering each primitive no matter whether it is visible. Using parallelization and efficient spatial data structures, ray tracing starts to be suitable also for interactive rendering [26, 27].

Given the success of point sets as a representation of shapes and ray tracing as a rendering technique it is natural to devise methods for ray tracing point sampled geometry. In particular, we use Point Set Surfaces (PSS) to define the geometry represented by the points [1, 2, 17]. This surface definition has several advantages in the context of ray tracing:

- The computation is local, which allows to evaluate the surface only in the vicinity of the ray-surface intersection.
- It is possible to define a minimum feature size, which can be exploited when computing the intersection of the surface with a ray.
- The surface is smooth and manifold, which makes CSG operations feasible.

Note that PSS require to compute a non-linear projection operator, which is a time consuming operation. Thus, the primary goal of this work is to minimize the number of projections. Due to the lengthy projection procedure ray tracing a PSS is far from being interactive. Moreover, other surface definitions based on point sets might lead to faster ray tracing procedures. The aim of our research is to present ray tracing techniques specifically suited to PSS because of their superior theoretical properties.

The definition of point set surfaces together with an explanation of the above mentioned properties is given in Section 3. The main contributions of this work are an efficient method to compute the intersection of a ray with the point set surface (Section 4) and several techniques to exploit coherence among neighboring ray-surface intersections (Section 5).

2 Related Work

For ray tracing point sampled geometry one may choose between two generally different approaches: Either the point set is used to compute a surface, which is ray traced,
or the intersection of a ray and the point set is defined without the intermediate definition of a surface.

Interestingly, the second alternative is not necessarily a surface definition. Schaufler and Jensen [25] define the intersection of a ray and the point set as follows: In each point a disk is constructed using the point normal. A cylinder around the ray is intersected with the disks. The intersection is computed as a weighted average of disks whose centers are inside the cylinder. Because the intersection point varies with the angle of the ray this procedure is not a surface definition.

Here we choose to use a surface definition because it is important for modeling and ray tracing:

- Primary and secondary rays intersect the same surface.
- The resulting image of the shape is view independent, which is a prerequisite for the generation of animated sequences.
- Renderings of CSG-defined shapes are possible.

However, PSSs are only one way to define a surface from points. Many such definitions have been developed, mainly as a first step in constructing explicit representations (e.g. triangle meshes or NURBS) from the points.

Hoppe et al. [14] define a signed distance field from the points. For each point a normal direction is estimated and the normal is oriented. The signed distance to the surface is defined as the normal component of the distance to the closest point.

In a sense the surface definition of Hoppe et al. is related to Voronoi-based reconstruction techniques [9, 3, 10, 5, 4] as the surface is defined by a closest point relationship. These techniques define the surface as a subset of the Delaunay triangulation of the point set. As such, they are all global methods and generate a $C^0$ surface.

Smoother surface definitions from points are typically implicit (e.g. [8, 20]). Note that also point set surfaces could be seen as an implicit surface.

Ray tracing an implicit surface is conceptually simple: The ray in its parametric form is substituted in the implicit surface definition. The problem of determining the intersection is, thus, equivalent to finding a root of a function in one unknown. The simplest approach to this problem is known as ray marching, i.e. searching for the root on a (possibly hierarchically refined) regular tiling of the ray.

To speed up the intersection computation one can exploit properties of the implicit function. A common way is to compute a (local) Lipschitz constant [16, 13]. This bound on the derivative allows to compute a conservative step width, i.e. a step on the ray that cannot change the inside/outside status. Another approach is to use interval analysis [19]. Root finding is a general problem in numerical analysis and less domain specific techniques could be found in textbooks (e.g. [22]).

3 Surface definition and features

We first give a brief definition of point set surfaces (the interested reader is directed to [2]). Then we discuss some features of the surface and its computation that are important for intersection with a ray.

Given points $p_i \in \mathbb{R}^3, i \in \{1, \ldots, N\}$ on or close to a surface $S$. The point set surface $S_P$ is implicitly as the set of points that project onto themselves using the following projection procedure.

To project a point $r$ a reference domain is computed, in which the surface is locally approximated by a bivariate polynomial (see Figure 1). The reference domain is determined by minimizing the weighted distance of points to a plane $H = \{x | \langle n, x \rangle - D = 0, x \in \mathbb{R}^3 \}, n \in \mathbb{R}^3, \|n\| = 1, D \in \mathbb{R}$. Assume $q$ is the projection of $r$ onto $H$, then $H$ is found by locally minimizing

$$\sum_{i=1}^{N} \langle \langle n, p_i \rangle - D \rangle^2 e^{-\|p_i - q\|^2 / h^2}$$

where $h$ defines the feature size. The local reference domain is then given by an orthonormal coordinate system on $H$ so that $q$ is the origin of this system. In this reference domain a bivariate polynomial $g$ is fitted to the points minimizing the squared distances in normal direction of $H$. The projection $\rho$ of $r$ onto $S_P$ is defined by the polynomial value at the origin, i.e. $\rho(r) = q + g(0,0)n$.

The above procedure of determining a minimum of Gaussian-weighted distances has two important implications:

![Figure 1. The projection of a point on the surface. First, a local approximating plane is computed, which serves as a reference domain. Second, a local bivariate polynomial approximates the surface within the reference domain. The projection is defined as the intersection of the polynomial and the orthonormal projection onto the reference plane.](image-url)
Figure 2. Point set surfaces allow to control the feature size with a scalar parameter, here called \( h \). A ball of radius \( h \) intersects the point set surface always in one connected component. This means that regions of higher curvature (as depicted in the right illustration) or two sheets closer than \( 2h \) are impossible.

Figure 3. Separation of features: If the distance between two points is larger than \( 2h \), they belong to potentially different components of the surface because their Gaussian weights could be separated (left illustration). If the distance is less than \( 2h \), the Gaussian weighted points necessarily belong to the same component (right illustration).

1. The projection operator works only in a tubular neighborhood around the point set. Points far from the point set are projected to infinity.
2. Local features are larger than \( h \). This means a ball of radius \( h \) intersects only one connected component of the surface (for an illustration, see figure 2).

The first property limits the use of the projection operator for the computation of ray-surface intersection to points on the ray that are close enough on the surface. As we will see in the next Section, this is not a problem.

The second property is an important feature for the design of the ray-surface intersection computation. It essentially means that inside a ball of radius \( h \), the polynomial approximation \( g \) is homotopy equivalent to the surface \( S_p \) inside that ball.

To understand this feature of the surface definition look at the univariate case of two shifted Gaussians of the form

\[ e^{-(x\pm d/2)^2/h^2} \]

which could be seen as a one-dimensional cut through a small region of a point set surface (see figure 3). If the sum of these Gaussians has one maximum, the points belong to the same surface; if the sum has two maxima, the points define distinct surfaces. It is easy to check that \( e^{-(x+d/2)^2/h^2} + e^{-(x-d/2)^2/h^2} \) has only one maximum for \( d \geq 2h \) and two otherwise. Thus, if a ball of radius \( h \) includes both points they belong to one surface. Note that this argument extends to surfaces only approximately because of irregular point-to-point distances. Nevertheless, we find the property to be satisfied for most practical data sets.

Finally, note that larger values for \( h \) necessarily lead to longer computation times for the projection operations. This means that, in contrast to common LOD techniques in interactive rendering, coarser approximations of the shape need longer to render.

4 Computing ray-surface intersections

The general idea of computing ray-surface intersections is to converge iteratively by projecting points from the ray onto the surface \( S_p \). Every projection of a point \( r \) provides following useful information about the relative location and orientation of \( S_p \):

- The distance \( d_r(r) = ||r' - r|| \) of \( r \) to its projection \( r' = \rho(r) \). As the projection is approximately perpendicular to the surface, \( d_r(r) \) is a reasonable estimate of the distance of \( r \) to the surface.
- The direction of the projection \( \frac{r' - r}{||r' - r||} \), which is approximately the direction to the surface.
- The local plane \( H \) of \( \rho \).  
- The local bivariate polynomial approximation of \( \rho \).

In each step the local polynomial of a projected point from the ray is intersected with the ray. Because the polynomial approximates the surface adequately within a neighborhood of the projection \( r' \), the intersection of ray and polynomial
is close to the intersection of ray and point set surface – assuming the projected point \( r_0 \) is in the neighborhood of the point set (see Section 3). To find the point on the ray \( r_0 \) that is projected first the ray is intersected with a union of spheres enclosing the point set surface. This process is detailed and explained in Section 5.

Figure 4 illustrates the method: a point \( r_i \) is projected onto \( r_i' \) (which is on the surface) providing a local polynomial approximation. The intersection of the ray with the polynomial results in a point \( r_{i+1} \) on the ray that is expected to be closer to the eventual intersection. This procedure is repeated until the required precision \( \epsilon \) is achieved, i.e. \( d_p(r) < \epsilon \). Note that in the intersection point the surface is equivalent with the polynomial, which is why we expect the procedure to terminate.

In general, computing the intersection of a ray and a polynomial patch could be quite difficult, depending on the degree of the polynomial [12]. To avoid iterative computations we simply use polynomials of degree 2.

### 4.1 Region of trust for local polynomials

Let \( s \) be the intersection of the ray and the surface. We assume that \( ||r - s|| > ||r' - s|| \), i.e. that the distance to the real intersection of the ray is decreases when intersecting with the local polynomial approximation. This assumption depends on the accuracy of the polynomial approximation, which naturally depends on \( ||q - s|| \approx ||r' - s|| \), i.e. the distance between ray-surface intersection and projected point on the ray.

It is clear that \( \rho(r) \) provides a reliable polynomial approximation only within sufficient proximity of \( r' \). It is necessary to restrict the neighborhood to a region of trust \( T \) with radius \( T_r \) around the point \( r' \). We have to chose \( T_r \) so that no intersection of the ray with the surface is ignored and so that ray-polynomial intersections always increases accuracy. Figure 5 illustrates the problem. While \( T_1 \) is a good choice, the use of \( T_2 \) results in a misinterpretation of the situation: the existing intersection of the ray with the surface is not detected.

Strictly speaking, ray-polynomial intersection is reliable only for \( T_r = 0 \). With growing \( T_r \) the risk of false decisions increases, in particular for strongly convoluted surfaces and rays approximately parallel to the tangent plane. Note, however, that the deviation of the surface from the polynomial approximation is naturally bounded in a ball by its radius. Within the set of balls of radius smaller than \( h \) this bound is a monotone function of the radius, because the surface is homotopy equivalent to the polynomial approximations (see Section 3). This property is sufficient for a convergent behavior of the intersection procedure, if each projected point is closer than \( h \) to the surface. Thus, it is natural to set the region of trust \( T_r \leq h \).

Even within this region there is a theoretical chance that a ray misses the polynomial approximation but intersects the surface. This is most likely for saddle points and rays almost tangent to the saddle. In practice we found no occurrences of this potential problem.

### 4.2 Enclosing sphere structure

From the discussion above it is clear that we can apply the iterative scheme only for points \( r \) on the ray that are close enough to the surface. More precisely, a ball of radius \( h \) around \( r \) has to intersect \( S_P \). Unfortunately, \( S_P \) is unknown a priori.

The only a priori knowledge are the points \( p_i \), which are expected to be very close to the surface \( S_P \). Thus, we construct a set of balls of radius \( h \) around the points \( p_i \). The intersection of the ray with the union of the balls is clearly in the region of trust of the local surface approximations.

While the intersection of ray and union of balls satisfies the proximity constraint it is also necessary that any ray that intersects \( S_P \) also intersects the union of balls. This is trivially satisfied, if the union of balls completely encloses the surface, i.e.

\[
S_P \subseteq \sum_{i=0}^{N} B_i.
\]  

Note that this has to be satisfied by virtue of the feature size \( h \): If the points leave a gap so that a ball is empty, this is a feature: namely a hole in the surface.

The intersection of a ray with the set of balls can be efficiently computed. These intersections provide the points of origin \( r_i \) for the application of the iterative scheme explained above. The refinement procedure is terminated once \( r_i \) is outside the ball or when the desired accuracy is achieved. If no ray-polynomial intersection is found inside one ball, other intersected balls are inspected.

In practice the radius of the balls \( B_r \) is chosen to be smaller than \( h \). Reducing the size of the balls minimizes the
distance of the starting point for the projection procedure and makes it more stable and faster. In addition, potentially less balls are intersected by the ray so that less balls have to be handled. While reducing the radius one has to make sure that the surface containment property 2 is satisfied. Note that this requires $B_r > h/2$ to avoid holes in the union of balls.

5 Implementation

In this section, several techniques to accelerate the computation of ray-surface intersections are discussed and the resulting algorithm is presented.

5.1 Spatial hierarchy

For the computation of a ray-surface intersection it is necessary to check all the spheres for potential intersections. To avoid unnecessary ray-sphere intersection tests we use a bounding sphere hierarchy that is build bottom-up from the region of trust spheres.

Usually it is favorable to traverse the hierarchy front to back with respect to the viewer. For such a traversal one has to maintain the closest ray-surface intersection point determined during the computations. However, as ray-sphere intersections are much cheaper than carrying out the iterative refinement procedure, it is preferable to first collect all the spheres intersecting the ray and sort them front to back with respect to the viewing direction. Then each sphere is inspected for a potential ray-surface intersection. We have found this approach to be more effective than computing intersections while traversing the bounding sphere hierarchy. In practice the ray surface intersection is typically found in the closest sphere that intersects the ray.

5.2 Exploiting coherence

When a ray intersects a sphere a point needs to be projected onto the surface to yield the local polynomial approximation. We always use the sphere center for the first projection (see Figure 6 for an illustration). This has two reasons:

1. The resulting polynomial approximation has a more equal error distribution w.r.t. the sphere as compared to off-the-center projection points.
2. The coefficients of the polynomial are stored and reused for the next ray intersecting this sphere.

When a ray intersects a sphere, oftentimes the surface inside the sphere is not intersected. In this case the ray typically does not intersect the polynomial approximation so that the refinement procedure terminates after one intersection test. If, on the other hand, the ray intersects the surface, not more than 2–3 intersections tests are needed to achieve the desired accuracy.

Primary rays are nearly parallel so that neighboring rays are likely to intersect the same spheres. In addition, secondary rays have to pass the the same spheres as the corresponding primary rays. Storing the result of the first projection avoids the recomputation for each intersecting ray. As the number of projections inside one sphere is only 1–3 this approach saves the majority of projection operations.

To quantify the gain of this strategy we have analyzed the primary rays in the ray tracings of the bunny model (see Figure 9). From a total number of 140724 projections 80232 could be reused. As expected, a miss usually detected after the first projection (1.02 average projections).
On the other hand, it is superfluous to save more than one polynomial per sphere: A single refinement step approaches the surface-intersection so quickly (see Section 6) that too many polynomials would have to be stored to find one in the vicinity of the intersection. Likewise, less than one polynomial per sphere (i.e. sharing polynomial approximation among several spheres) is impossible, because this violates the region of trust concept.

5.3 Sorting for shadow rays

Once a shadow-ray is obstructed by an opaque object, it is not necessary to determine at what position the ray first hits that object. Such a ray can be simply discarded from the illumination computations. Therefore shadow ray intersection can be optimized by finding an intersection as quickly as possible anywhere on the ray.

In the shadow areas of an object the rays often intersect many hulls. Figure 7 illustrates such a case. The inner ray hits a lot of spheres, penetrating the surface twice. As spheres intersected close to their center are more likely to contain an intersection with the surface, they are sorted according to the distance $d$ from the shadow-ray to the center $B_c$. The following equation determines the priority $p_B$ of a sphere considering different radii $B_r$.

$$p_B = \frac{B_r - d}{B_r}.$$

5.4 Algorithm

The above discussion results in the following ray-surface intersection algorithm:

1. Traverse the spacial hierarchy, collect all spheres intersected by the ray.
2. Sort the spheres with respect to their distance from the ray-origin. In case of a shadow-ray, equation 3 determines the priority.
3. Pop a sphere from the priority-queue and execute the refinement procedure:

   (a) Check if the coefficients of the original polynomial approximation are already computed. If so, go to 3b, using the data. Otherwise project the sphere-center $B_c$ and store the information for further use.
   (b) Calculate the ray-polynomial intersection, making it the current reference $r_i$. If the projected distance $d_p(r) < \epsilon$, the ray-surface intersection is found. If no intersection is found inside $B_r$, proceed with 3.
   (c) Project $r_i$ and go on with 3b.

6 Results

We have evaluated our implementation on synthetic and real world data. The main performance criteria are speed and accuracy. Since the projection operator is a primitive operation that is given by the definition of PSS, we have to measure efficiency in terms of projections per intersection. Note that projection operations easily dominate the computation time. To quantify this, we have analyzed the primary rays for the ray trace of the bunny model (see Figure 9). The total computation time is 232.27 seconds, of which 226.37 seconds were used for a total of 140724 projection operations (3.7 msec on average per projection). Timings were acquired on a mobile P3 1MHz. Note that we have used an existing implementation of the projection operation and that the bounding hierarchy traversal is unoptimized.

For the estimation of accuracy, synthetic models are used because the PSS of arbitrary shapes cannot be determined a priori. As a synthetic test we have generated a dense point set on a sphere by subdividing an icosahedron. For a unit sphere a maximum error of $\epsilon = 10^{-13}$ was set. This bound was satisfied for any random ray in 2 or 3 iterations. Note that we could not evaluate the behavior for a smaller bound since the point set surface naturally deviates slightly from the analytical sphere.

For checking the behavior on real world data we generated images using different kinds of input: As a relatively regular input vertices from triangle meshes were used (see Figures 8 and 9). In addition we used points coming directly from a scanning device. This data represents points coming from consolidated depth images. The result is depicted in Figure 10.

Note that for real world data it makes sense to look at the
precision and convergence in terms of the feature size $h$. In general an intersection is computed to a specified precision of $10^{-6} h$ in about 2–3 iterations. For the particular example of the bunny, an average of 2.74 projections was needed to compute the intersection with the surface to the specified accuracy, while a miss was detected with an average of 1.02 projections. With each projection and intersection with the polynomial approximation the distance to the final intersection is typically reduced by a factor of $10^{-4} h - 10^{-6} h$.

An absolute precision of $10^{-13}$ was always possible to achieve, however, beyond that bound we could not see a stable increase in accuracy. We suspect this bound on the absolute accuracy to be based on the complex non-linear optimization procedure of the projection operator. Nevertheless, the resulting images are very smooth, even if the point information is sparse (see Figure 9).

7 Conclusions

We have presented a method to efficiently compute intersections of a ray and PSSs. Furthermore, several techniques to exploit spatial coherence among the ray-surface intersections were introduced. The implementation of the procedures allows to generate realistic images of PSS. The computation time is dominated by the non-linear projection operator (approximately 98%) – this means improving the computational speed of the projection operation would directly improve our ray tracing procedures. We are currently looking into approximate projection procedures that would lead to an tremendous speed-up.

The approach could be applied to other surface definitions, provided they have a bound on the feature size and allow to compute local polynomial approximations. Note that for rendering raster images a bound on the feature size is very natural as features smaller than a pixel are lost in the rendered image, anyway.

8 Acknowledgments

The bunny and the dragon model are courtesy of the Stanford Computer Graphics Laboratory. The afarensis model was 3d-digitized by Peter Neugebauer of Polygon Technology Ltd, Darmstadt, Germany using a structured light scanner and the QTSculptor system.

References

Figure 10. Results of the ray tracing procedure applied to raw point data. The afarensis (early human) model consists of 150,000 points acquired with a structured light scanner from a physical model.


