Ray Tracing Point Sampled Geometry

Gernot Schaufler
Massachusetts Institute of Technology

Henrik Wann Jensen
Stanford University

Abstract. We present a novel technique for ray tracing geometry represented by points. Our approach makes it possible to render high quality ray traced images with global illumination using unstructured point–sampled data thus avoiding the time-consuming process of reconstructing the underlying surface or any topological information. Compared with previous point rendering methods, our approach allows for more complex illumination models while still maintaining the simplicity of the point primitive.

Intersections with the point–sampled geometry are detected by tracing a ray through the scene until the local density of points is above a predefined threshold. We then use all the points within a fixed distance of the ray to interpolate the position, normal and any other attributes of the intersection. The considered distance from the ray must be larger than the largest "hole" among the points.

We demonstrate results for soft shadows, reflection and refraction, global illumination and subsurface scattering.

Keywords: points, ray tracing, global illumination

1 Introduction

As geometry is getting more complex, the triangles – today's most popular modeling primitives – are getting smaller and smaller. Soon, the overhead associated with them will no longer be justified, given that they only occupy sub-pixel areas in image space. The alternatives explored currently in the research community are higher order surfaces or simpler primitives such as points.

Points became popular in particular with the introduction of particle systems [17] but have also been used by rendering systems as the final target for subdividing more complex modeling primitives [4]. Today they have seen a comeback in image–based rendering and real-time graphics due to their modeling flexibility, efficiency, and ease of acquisition with digital cameras, 3D scanners and range finders,

Given the simplicity of points and the growing complexity of the geometric models, it seems natural that point geometry will become an important element in modeling and rendering. It is therefore desirable to extend the available rendering algorithms to the realm of photo–realistic image synthesis and physically–based lighting simulation.

This paper extends previous rendering approaches by introducing a method for ray tracing point–sampled geometry. This enables more complex illumination models and even makes it possible to simulate global illumination in scenes containing point sampled geometry. We have developed an intersection technique that uses only a local sampling of the point sampled geometry. This makes it possible to skip the time consuming surface reconstruction step and instead make high–quality renderings directly from the points representing the geometry.

2 Previous Work

To date, direct rendering of points has mostly proceeded in a "forward" fashion using the perspective transformation to map 3D points to 2D image locations. The image-based rendering literature is rich in techniques how to deal with the fact that with this approach not necessarily every output pixel receives a sample. Pioneering work [11] has been done by Levoy and Whitted in 1985, in which they propose points as a universal modeling primitive and present algorithms allowing for anti-aliased rendering of this representation. Cook *et al.* [4] propose to decompose more elaborate primitives to points in image space during rendering. More recent approaches include work by Grossman *et al.* [7], Pfister *et al.* [16], and Rusinkiewicz *et al.* [18]. The differ by how visibility among points is established and how the final image is reconstructed.

In image-based rendering, points are usually organized either into images with depth or layered–depth images [19]. For those images, incremental algorithms can be devised which make use of this particular organization of the points into 2D grids. Chen *et al.* [2] approximate the apparent motion of 3D points using optical flow. Dally *et al.* [5] present a method to build hierarchical object representations from points. Shade *et al.* [19] introduce layered depth images to obtain a complete sampling of the objects rather than just a sampling of the visible surfaces. A recent approach [14] factors the warping equation into two steps simplifying the reconstruction.

Somewhat related to layered-depth images of 3D geometry are volume representations. These have been rendered using "splatting" of individual volume elements as in [25, 10, 12] or with hardware acceleration [24].

All these approaches have one thing in common: no global illumination models can be applied to the geometry. In many cases the color stored per sample is copied into the final image directly. The only global effect reported so far is sharp shadows from point lights using shadow maps [15].

3 Point Sampled Geometry

The most popular sources of point–sampled geometry are 3D scanners and laser range finders. While many scans are usually required to cover the surface of complex objects with samples, there exist efficient methods such as the one by Chen *et al.* [3] and Nyland *et al.* [13] which register multiple scans into a common coordinate system without explicitly reconstructing the surfaces in the scene. In [13] planes are fit to points on known planar surfaces which are aligned between different scans. This approach works best for indoor scenes, where samples on walls are aligned. For scans of smaller objects, background planes can be added to the scanned geometry artificially. We build on the success of such methods and assume registered point samples of complex objects to be available.

In order to unambiguously represent a surface by a sampling of points, this sampling must fulfill a number of requirements in order to distinguish close but different surface portions from each other, and to distinguish holes in the surface from regions of sparse sampling. In particular, the sampling density of the points must be at least twice as high as the minimum distance between different portions of the surface [8], *i.e.* at least as high as the distance to the nearest point on the medial axis of the surface [1].

Our rendering approach assumes that the maximum size of a gap in the samples is known. If the point samples are not that uniformly distributed over the surface, they can be made to be more evenly spaced using an approach described by Szeliski *et al.* [21, 22]. By giving attracting and repulsive forces to the points, points tend to even out the spacing between them. Adding points at gaps along borders can fill holes of a given maximum size.

Like Szeliski *et al.*, we also assume a normal to be available per point. If the 3D scanning process does not provide normal information, it can be estimated by fitting a plane to the points in the neighborhood of each point using least-squares minimization. The nearest neighbor search can be accelerated using spatial data structures such as octrees or bsp-trees, giving normal computation a complexity of $O(n \log n)$ [8]. Note that we do not reconstruct a surface as Hoppe *et al.*but only compute normals from fitting tangent planes to each point. Our implementation is capable of pre-computing on the order of 5000 normals/second for geometry represented by 440000 points on a MIPS R10k processor running at 250MHz. Alternatively, normals can be computed on the fly as intersections are detected.

4 Intersecting a Ray with Point Geometry

Computing an intersection with the point geometry is split into two parts: detecting if an intersection has occurred and computing the actual point and normal of the intersection.

4.1 Intersection Detection

An intersection is reported, if the ray intersects a disk centered at each point. These disks have the same normal as the surface point. The radius r of the disks is set slightly bigger than the radius of the biggest hole in the points' sampling of the surface. We accelerate the search for such a point using an octree. The ray is surrounded by a cylinder of radius r and only those nodes in the tree that intersect the cylinder are traversed. The cylinder is capped off at the origin of the ray and the maximum length of the ray using two planes. Further acceleration is achieved by projecting the cylinder onto two coordinate planes and checking the octree nodes with these projections (see Figure 1).

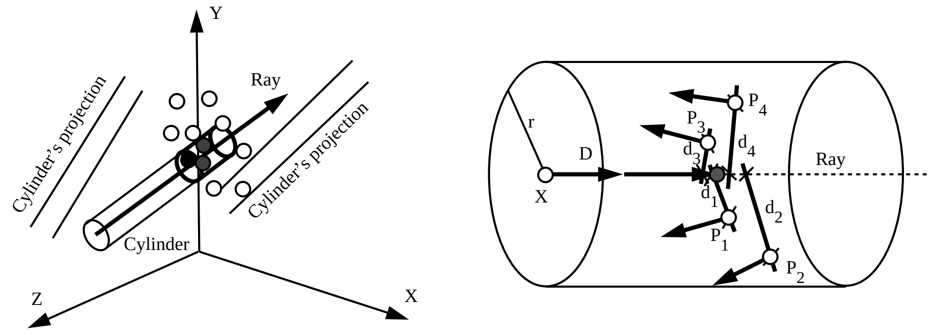


Fig. 1. Left: Cylinder around ray with projections onto coordinate planes. The first disk centered at a point intersected by a ray (black) triggers an intersection. Position, normal and other per-point attributes are interpolated for all points inside the bold cylinder (grey) starting at the first found point. Right: Calculating the intersection point (grey) from four points in the cylinder. Their normals and positions along the ray are weighted by their distance d_i from the ray.

The two planes are chosen based on the largest component of the ray's direction vector. The example in the figure shows a ray with the Y-coordinate largest. Therefore, the cylinder is projected onto the XY- and YZ-planes.

4.2 Intersection Point Computation

Once an intersection is detected, another short cylinder is started at this point along the ray (shown in bold on the left of Figure 1) and all the points within this cylinder are collected. The actual intersection point will be interpolated from these points. We have found our simple interpolation scheme to give visually satisfactory results although higher order interpolation is possible. In particular, the normal of the intersection point is formed by weighting the normals of the points in the cylinder based on the point's distance from the ray. The same weighting is applied to find the actual intersection point along the ray. We intersect the planes formed by each point and its normal with the ray's parametric representation, $ray = X + t * D$, where X and D are the origin and the direction of the ray. The final intersection point is found by interpolating the computed t values, and inserting this value in the ray's parametric representation.

The right of figure 1 shows a 2D example of a ray running through a group of four points. For each point P_i, the normal and local plane are shown which intersect the ray's supporting line at a distance d_i from the point. The normal of the intersection point (shown in grey) is then calculated as in equation (1). In general, any attribute associated with the points can be interpolated this way.

$$attrib = \frac{\sum_i attrib_i * (r - d_i)}{\sum_i (r - d_i)} \tag{1}$$

Points will only be considered if their distance from the ray d_i is smaller than r. Note that the distance is *not* measured orthogonal to the ray but in the plane of the point and its normal to create the circular disks around each point.

This intersection point computation is slightly view dependent. The final position of the surface will vary with the direction of the incoming ray. The variations are small enough that they do not result in visible changes to the surface, but for more complex illumination algorithms this discrepancy must be taken into account as explained in the next section.

5 Rendering with Points

Given a technique for intersecting a ray with points, we can apply more advanced illumination models. We also augment each point with extra information such as texture coordinates, color, diffuse albedo etc. that can be interpolated and used by the ray tracer. Recursive ray tracing considers shadows and reflected and transmitted light. Furthermore, we can use Monte Carlo ray tracing methods to compute global illumination.

As mentioned in the previous section, the computed intersection point is slightly dependent on the direction of the incoming ray. Even though this does not result in visible changes to the surface, it must be taken into account within the ray tracer. This is particularly important for shadow rays where wrong ray-geometry intersections can result in false self-shadowing of the surface. To avoid this problem, we use a shadow offset based on the radius r of the disks. We have found a shadow offset of 1-3 times r to work well in practice. We do not consider this to be a problem of using points since shadow offsets are necessary even for triangle meshes with vertex normals [20]. In addition to the shadow offset we only accept intersections with point–sampled geometry that are at least $2r$ away from the ray origin. This is to prevent wrong intersections for reflected and transmitted rays.

Normals are either computed on the fly or in a pre-processing step. These normals can possibly point into the interior of objects. To prevent this from being a problem in the ray tracer, we flip the point normals such that they always point towards the ray origin, meaning we cannot use the normal to determine whether the ray enters or leaves a given object. For this purpose we use the history of the ray - ie. by counting the number of previous transmissions we can determine whether we are inside or outside a given object.

6 Results

We have implemented the intersection technique for point sampled geometry as a geometry component in our ray tracer. This ray tracer also supports global illumination via stochastic sampling of indirect light optimized with irradiance caching [23] and photon maps [9]. All timings and results were obtained using a dual PII-400MHz.

Our first sequence of images in Figure 2 illustrates how the radius of the cylinder around the ray is approaching the distance between the points. Eventually, the single points join together into a smooth surface. Bigger radii result in more smoothing.

We have also implemented texture coordinate interpolation as another example of a surface attribute specified per point. Figure 3 gives two examples of a texture mapped onto geometry specified as points. Interpolation of texture coordinates in between points is accomplished using Equation 1 from Section 4.

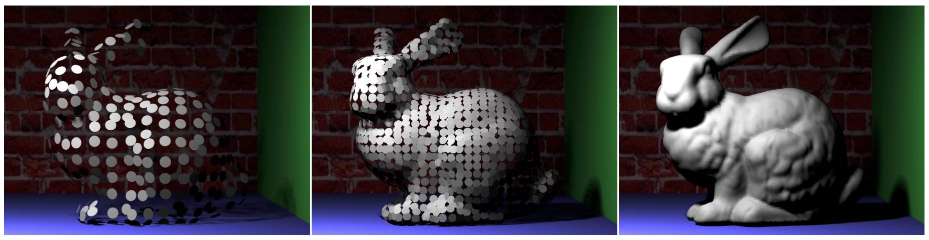
The image in Figure 4 demonstrates global illumination and caustics on point sampled geometry. It is a glass bunny generating a caustic on a wooden bunny. The rendering time for this image was 11 minutes given a resolution of 768x512 and 4 samples per pixel.

Our final example in Figure 5 illustrates our approach on a complex model. It is a scanned version of the head of Michelangelo's David statue. The head has more than two million points. Each point has an additional diffuse albedo attached.

Figure 5(a) shows the head rendered with global illumination and using the interpolated diffuse albedo information as the local surface color. This model was rendered in 1024x1024 with 4 samples per pixel in 61 minutes. For this model we optimized the size of the radius such that the surface just connects, and so we are able to capture every fine detail of the model. We precomputed the normals using the 3-6 nearest neighboring points.

Figure 5(b) shows an example of a complex translucent illumination model applied to the head. We rendered the head using subsurface scattering with multiple scattering [6] based on highly translucent artificial volumetric marble. Notice how the point sampled surface provides enough information to perform this simulation. The image does have a few artifacts in areas with holes in the model (such as in the nose and in the hair), but the overall appearance is captured faithfully. The image was rendered in 1024x1024 with 4 samples per pixel in approx. 4 hours.

To test the speed of our point intersection code we compared it with triangles in a number of simple test scenes: one containing the bunny and one containing the Buddha appearing in the video. Table 1 shows the resulting timings. The points code has not yet been optimized, and it can be observed from the table that our optimized triangle intersection code is approximately 3-4 times faster. For the triangle meshes we used a hierarchical grid. We believe that most of this overhead is due to traversing the octree.



(a) Bunny: 500 points, r = 0.04 (b) Bunny: 1900 points, r = 0.03 (c) Bunny: 34800 points, r = 0.014

Fig. 2. Rendering a bunny with a variety of point counts and cylinder radii.

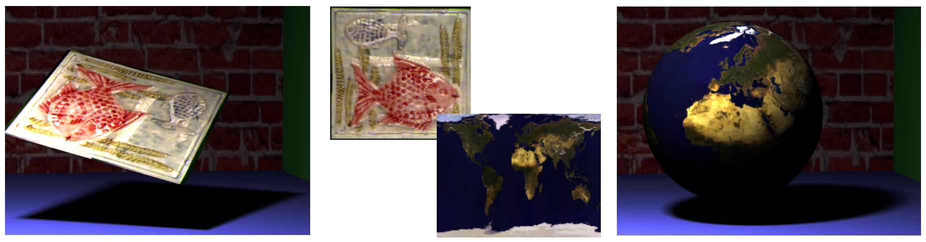


Fig. 3. Interpolation of texture coordinates in between points.

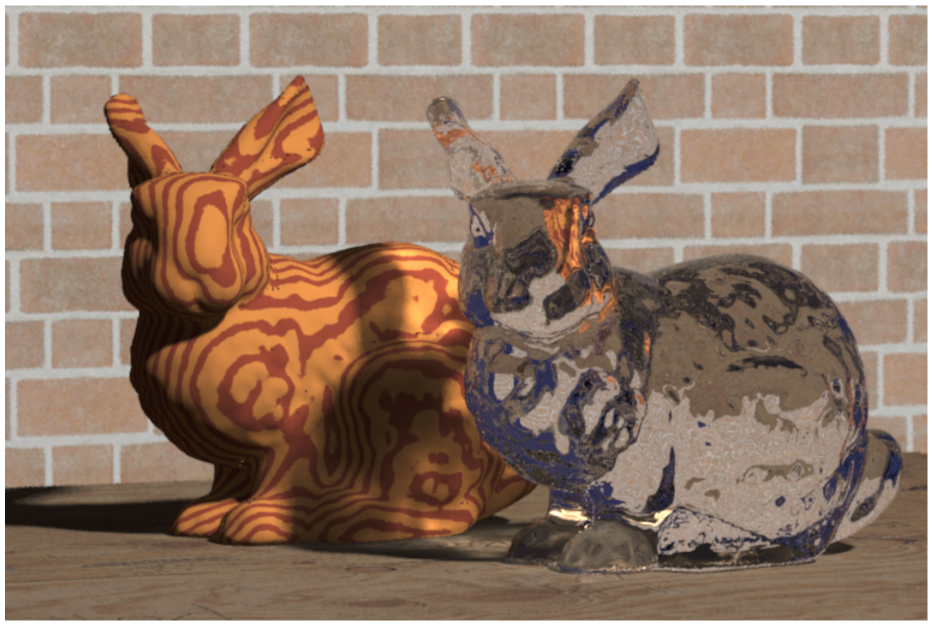


Fig. 4. A caustic from a glass bunny onto a wood bunny.

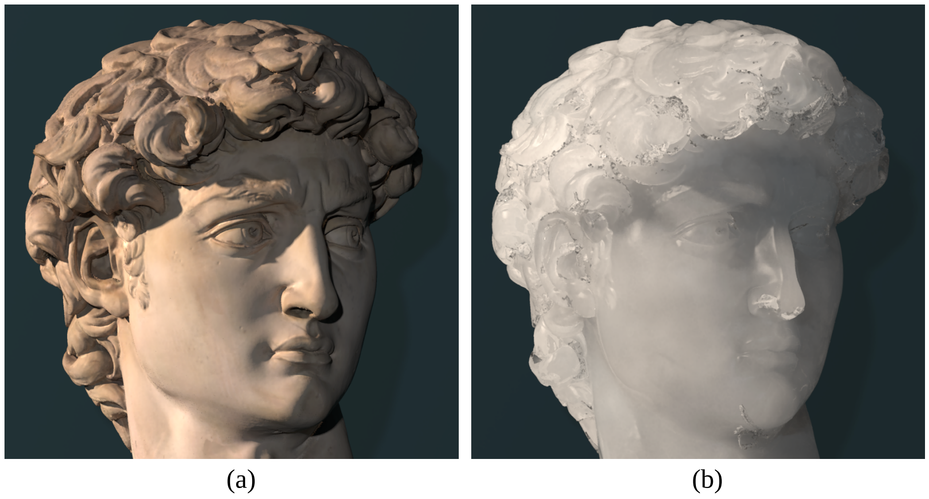


Fig. 5. The head of the David rendered from 2 million points. (a) Global illumination using points with color information, (b) Subsurface scattering using artificial volumetric marble.

Model	Points/Triangles	Render time (sec.)	Rays/sec.
Bunny (points)[a]	34834	23.1	34204
Bunny (points)[b]	34834	24.5	32142
Bunny (triangles)	69451	8.4	93809
Buddha (points)[b]	543652	46.1	12309
Buddha (points)[c]	543652	36.1	15731
Buddha (triangles)	1087716	8.6	63300

[a]5 octree levels
[b]6 octree levels
[c]7 octree levels

Table 1. Intersection timing (points vs. triangles).

7 Discussion

In the process of trying to obtain these results we considered a number of simpler intersection techniques that did not work quite as well as the method we decided to use.

We tried the following alternative approaches:

- **Using a sphere around each point.** This method grows the object by the radius of the spheres. It requires a large number of spheres to make the surface appear mostly smooth – too few spheres will make the surface bumpy. The normals generated at intersection points are determined by where each sphere is hit and exhibit discontinuities between spheres.
- **Using an octree with non–cube shaped leaf nodes.** Within each leaf node the local subset of points can be used to construct a small surface patch (a plane or a higher order surface) which will possibly cause an intersection with the ray in this part of space. This technique suffers from lack of continuity between the leaf nodes. Moreover, the border of the patch is determined by the faces of the octree node. In many cases the patch will inappropriately extend into the corners of nodes.
- **Using an oriented disc at each point.** This approach is closest to what we currently apply for ray intersection, but without interpolation between points, it will not give a smooth surface. As other attributes are not interpolated, normals or color per point and texture mapping will not work as expected. The disks will appear as flat-shaded patches.

In our approach we use a cylinder to locate the points from which attributes are to be interpolated. This results in a visually smooth surface – its smoothness can be controlled by varying the disk radius. As pointed out, our definition of the ray-point geometry intersection causes the obtained surface to be slightly view-dependent. The interpolation of the ray-parameter t, from which the exact point of intersection is derived, will be different for different directions of incidence of the ray. However, in the animations we have rendered, this did not cause any visually distracting artifacts. We attribute this to the fact that the shading of the surface is mostly determined by the normal, which is unaffected by the exact location of the intersection.

An interesting question is how direct use of point sampled geometry compares to using triangles. As our timings statistics in table 1 indicate our intersection code for point sampled geometry is approximately 3-4 times slower than our optimized triangle intersection code. This is not bad considering that the points intersection code is the first rough implementation which has not been optimized. We believe that the point-geometry intersection is no more complicated than triangles since the disk test is much simpler than the barycentric test used for triangles. In addition most complex models have fewer points than triangles (the bunny has approximately 70000 triangles but only 35000 points). Memory usage is similar for the two methods; points use slightly less memory since they do not require connectivity information.

Another advantage with points is that it is easy to make hierarchical simplifications by merging neighboring points. As demonstrated in a recent splatting approach [18] it is possible to build a highly efficient hierarchical representation of point sampled geometry for use with very complex models (more than hundred million points). The complete model of the David has on the order of 1 billion points. A hierarchical point sampled representation may be the most efficient way to deal with a model of this complexity. Since we already use an octree to locate the points for our intersection computation is should be straightforward to test this concept.

Currently, we use a fixed radius to locate neighboring points. This assumes a similar density of points over the surface. For some objects it might be advantageous to wary the density of the points to capture certain local variations such as edges in the geometry more efficiently. For this purpose we would like to include an adaptive radius where the local density of points is used to adjust the size of the region from which points are used.

Our intersection routine for points is slightly view dependent. This has not caused problems in the models that we have rendered, but it would be nice to have completely consistent geometry. We have considered using a fixed direction for the cylinder that collects points (for example based on the normal of the first disk that is intersected). Another alternative would be to collect points using a sphere around the first intersection point.

8 Conclusion and future work

In this paper we demonstrate how global illumination effects can be achieved on geometry represented only by a sampling of 3D points. We formulated a method to intersect a ray with such a surface representation and smoothly interpolate surface attributes across it. By that we have extended the usefulness of this type of object representation to the field of realistic image synthesis and physically-based light transport.

In the future we would like to compute a more accurate surface intersection based on an adaptive local sampling of the points. In addition, we would like to make better use of our octree hierarchy and sample the geometry at a level which reflects the level of detail at which the point sampled geometry is observed. This could be done by storing filtered attributes at the interior nodes of the point hierarchy using a filtering similar to mip-mapping.

Even though it is still faster to use triangles for rendering our scanned models we believe that direct ray tracing of point sampled geometry has significant potential in particular as our models become more complex.

9 Acknowledgments

Thanks to Szymon Rusinkiewicz and Justin Legakis for helpful comments on the paper. The dataset of David's head appears courtesy of Marc Levoy and is part of the ongoing Digital Michelangelo Project. Other models appear thanks to their availability in the Stanford 3D Scanning Repository. This work was funded by DARPA DABTB63-95-C0085.

References

1. Nina Amenta, Marsahll Bern and Manolis Kamvysselis: “A new Voronoi-based surface reconstruction algorithm”, *Proc. SSIGGRAPH '98*, pp 415-421, 1998.
2. E. Chen and L. Williams: “View Interpolation for Image Synthesis”, *Proc. SIGGRAPH '93*, pp 279-288, 1993.
3. Yang Chen and Gérard Medioni: ”Object Modeling by Registration of Multiple Range Images”, *Proceedings of the International Conference on Robotics and Automation*, 1991.
4. Robert L. Cook, Loren Carpenter and Edwin Catmull: “The Reyes Image Rendering Architecture”, *Proc. SIGGRAPH '87*, pp 95-102, 1987.

5. William J. Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs: “The Delta Tree: An Object-Centered Approach to Image-Based Rendering”, *MIT AI Lab Technical Memo 1604*, May 1996.
6. Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis and Hans Køhling Pedersen: “Modeling and Rendering of Weathered Stone”, *Proc. SIGGRAPH ’99*, pp 225-234, 1999.
7. J. P. Grossman and W. Dally: ”Point Sample Rendering”. *Rendering Techniques ’98*, Springer, Wien, Vienna, Austria, pp 181-192, 1998.
8. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle: “Surface reconstruction from unorganized points”, *Proc. SIGGRAPH ’92*, pp 71-78, 1992.
9. Henrik Wann Jensen: “Global illumination using photon maps”. *Rendering Techniques ’96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, Springer Verlag, pp 21-30, 1996.
10. David Laur and Pat Hanrahan: “Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering”, *Proc. SIGGRAPH ’91*, pp 285-288, 1991.
11. M. Levoy and T. Whitted: “The Use of Points as Display Primitives”, *Technical Report TR 85-022*, The University of North Carolina at Chapel Hill, Department of Computer Science, 1985.
12. K. Mueller and R. Yagel, “Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach”, *Visualization ’96*, pp 65-72, 1996.
13. Lars Nyland, David McAllister, Voicu Popescu; Chris McCue and Anselmo Lastra: “Interactive exploration of acquired 3D data”. *Proceedings of SPIE Applied Image and Pattern Recognition Conference (AIPR99)*, Washington DC, October, 1999.
14. Manuel M. Oliveira, Gary Bishop, David McAllister: “Relief Texture Mapping”. *to appear in Proc. SIGGRAPH 2000*, July 2000.
15. Manuel M. Oliveira and Gary Bishop: “Dynamic Shading in Image-Based Rendering”. *UNC Computer Science Technical Report TR98-023*, University of North Carolina, May 31, 1998.
16. Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar and Markus Gross: “Surfels: Surface Elements as Rendering Primitives”, *to appear in Proc. SIGGRAPH 2000*, July 2000.
17. W. T. Reeves and R. Blau: “Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems”. *Proc. SIGGRAPH ’85*, pp 313-322, 1985.
18. Szymon Rusinkiewicz and Marc Levoy, “QSplat: A Multiresolution Point Rendering System for Large Meshes”, *to appear in Proc. SIGGRAPH 2000*, July 2000.
19. J. Shade, S. Gortler, L. He, and R. Szeliski: “Layered Depth Images”, *Proc. SIGGRAPH ’98*, pp 231-242, 1998.
20. John M. Snyder and Alan H. Barr: “Ray Tracing Complex Models Containing Surface Tessellations”, *Proc. SIGGRAPH ’87*, pp 119-128, 1987.
21. Richard Szeliski and David Tonnesen: “Surface modeling with oriented particle systems”, *Proc. SIGGRAPH ’92*, pp 185-194, 1992.
22. Richard Szeliski, David Tonnesen and Demitri Terzopoulos: “Modeling surfaces of arbitrary topology with dynamic particles”. *IEEE CVPR 1993*, pp 82-87, 1993.
23. Greg Ward, Francis M. Rubinstein, and Robert D. Clear. “A Ray Tracing Solution for Diffuse Interreflection”. *Proc. SIGGRAPH ’88*, pp 85-92, 1988.
24. Rüdiger Westermann and Thomas Ertl: “Efficiently using graphics hardware in volume rendering applications”, *Proc. SIGGRAPH ’98*, pp 169 - 177, 1998.
25. L. Westover: “Footprint Evaluation for Volume Rendering”, *Proc. SIGGRAPH ’90*, pp 367-376, 1990.