Point Sample Rendering

J.P. Grossman[1] and William J. Dally[2]

Abstract

We present an algorithm suitable for real-time, high quality rendering of complex objects. Objects are represented as a dense set of surface point samples which contain colour, depth and normal information. These point samples are obtained by sampling orthographic views on an equilateral triangle lattice. They are rendered directly and independently without any knowledge of surface topology. We introduce a novel solution to the problem of surface reconstruction using a hierarchy of Z-buffers to detect tears. Our algorithm is fast and requires only modest resources.

1 Introduction

The research presented in this paper was motivated by a desire to render complex objects in real time without the use of expensive hardware. This ubiquitous goal has recently led to the use of images, rather than polygons, as primitives. Unlike traditional polygon systems in which rendering time is proportional to scene complexity, in image based systems rendering time is proportional to the number of pixels in the source images or the output image. The techniques of view interpolation [Chen93, Chen95], epipolar geometry [Laveau94], plenoptic modeling [McMillan95], light fields [Levoy96], lumigraphs [Gortler96], view based rendering [Pulli97], nailboards [Schaufler97] and post rendering 3D warping [Mark97] have all demonstrated the speed and viability of image based graphics. Despite the success of these approaches, however, they suffer from a large appetite for memory, noticeable artifacts from many viewing directions, and an inability to handle dynamic lighting. In [Wong97] the light field system was extended to support dynamic lighting, but only at a significant cost in memory and with an unspecified degradation of rendering speed.

Point Sample Rendering is an algorithm which features the speed of image based graphics with quality, flexibility and memory requirements approaching those of traditional polygon graphics. Objects are modeled as a dense set of surface point samples which are obtained from orthographic views and stored with colour, depth and normal information, enabling Z-buffer composition, Phong shading, and other effects such as shadows. Motivated by image based rendering, point sample rendering is similar in that it also takes as input existing views of an object and then uses these to synthesize novel views. It is different in that (1) the point samples contain additional geometric information, and (2) these samples are *view independent* - that is, the colour of a sample does not depend on the direction from which it was obtained.

Points have often been used to model 'soft' objects such as smoke, clouds, dust, fire, water and trees [Reeves83, Smith84, Reeves85]. The idea of using points to model solid objects was mentioned nearly two decades ago by Csuri et. al. [Csuri79] and was briefly investigated by Levoy and Whitted some years later for the special case of continuous, differentiable surfaces [Levoy85]. More recently Max and Ohsaki used point samples, obtained from orthographic views and stored with colour, depth

[1] Dept. of EECS, Massachusetts Institute of Technology. Email: jpg@ai.mit.edu
[2] Department of Computer Science, Stanford University. Email: billd@csl.stanford.edu

and normal information, to model and render trees [Max95]. However, almost all work to date dealing with a point sample representation has focused on using it to generate a more standard one such as a polygon mesh or a volumetric model [Hoppe92, Turk94, Curless96].

The fundamental challenge of point sample rendering is the on-screen reconstruction of continuous surfaces. Surfaces rendered using point samples should not contain holes (figure 1). Furthermore, this issue should be addressed without seriously impacting rendering speed. The algorithm presented in [Levoy85] makes use of surface derivatives and is therefore applicable only to differentiable surfaces. In [Max95] no attempt is made to address this problem at render time, though it is alleviated by sampling the trees at a higher resolution than that at which they are displayed.

In this paper we will present a novel solution to problem of surface reconstruction which is critical to the success of our rendering algorithm. To obtain a more uniform sampling, orthographic views are sampled on an equilateral triangle lattice rather than a traditional square lattice. A pyramid of Z-buffers is used to detect gaps in the output image; these gaps are filled in screen space as the final rendering step. The use of a hierarchical Z-buffer incurs very little computational overhead, and the screen space operation of filling gaps is quite fast.

We have implemented a software test system to demonstrate the capabilities of point sample rendering. The system takes as input object descriptions in Open Inventor format and produces point sample models as output. These models are then displayed in a window where they may be manipulated (rotated, translated and scaled) in real time by the user.

2 Surface Reconstruction

Given a set of surface point samples, it is easy enough to transform them to screen space and map them to pixels. When a point sample is mapped to a pixel, we say that the pixel is *hit* by the point sample. When a pixel is hit by points from multiple surfaces a Z-buffer can be used to resolve visibility. The difficulty is that, in general, not all pixels belonging to a surface will be hit by some point from that surface. This leaves holes through which background surfaces are visible (figure 1).

One approach to this problem is to treat the point samples as vertices of a triangular mesh which can be scan-converted. However, this method is extremely slow, requiring a large number of operations per point sample. Another approach is the use of 'splatting', whereby a single point is mapped to multiple pixels on the screen, and the colour of a pixel is the weighted average of the colours of the contributing points. This method was used by Levoy and Whitted [Levoy85]; it has also been used in the

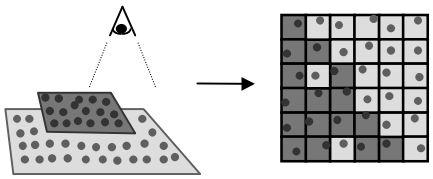


Figure 1: A dark surface is closer to the viewer than a light surface. When the surfaces are rendered using point samples, the points on the dark surface are more spread out. Some screen pixels are 'missed', and the light surface peeps through.

context of volume rendering [Westover90]. However, this is also too slow for our needs, again requiring a large number of operations per point.

In order to maximize speed, our solution essentially ignores the problem altogether at render time, as in [Max95]. To see how this can be achieved without introducing holes into the image, we start with the simplifying assumptions that the object will be viewed orthographically, and that the target resolution and magnification at which it will be viewed are known in advance. We can then, in principle, choose a set of surface point samples which are dense enough so that when the object is viewed there will be no holes, independent of the viewing angle. We say that an object or surface is *adequately sampled* (at the given resolution and magnification) if this condition is met. To fashion this idea into a working point sample rendering algorithm, we need to answer the following three questions:

- How can we generate an adequate sampling using as few points as possible?
- How do we display the object when the magnification or resolution is increased?
- How must we modify the algorithm to display perspective views of the object?

These questions will be addressed in the following sections.

2.1 Geometric Preliminaries

Suppose we overlay a finite triangular mesh on top of a regular array of square pixels. We say that a pixel is *contained* in the triangular mesh if its center lies within one of the triangles (figure 2a). As before, we say that a pixel is *hit* by a mesh point if the mesh point lies inside the pixel.

Theorem If the side length of each triangle in the mesh is less than the pixel side length, then every pixel which is contained in the mesh is hit by some mesh point.

Proof Our proof is by contradiction - suppose instead that some pixel P is contained in the mesh but is not hit by any mesh point. Then there is some mesh triangle ABC such that ABC contains the center of P, but A, B, C lie outside of P. Now we cannot have one vertex of ABC above (and possibly to the left/right of) P and another below P, as then the distance between these vertices would be greater than the side length of P (figure 2b). Similarly, we cannot have one vertex to the left of P and another vertex to the right of P. Without loss of generality, then, assume that A, B, C all lie to the left of and/or below P (figure 2c). The only way for a triangle with vertices in this region to contain the center of P is if one vertex is at the upper left corner of P and the other is at the lower right - but then the distance between these vertices is greater than the side length of P, contradiction.

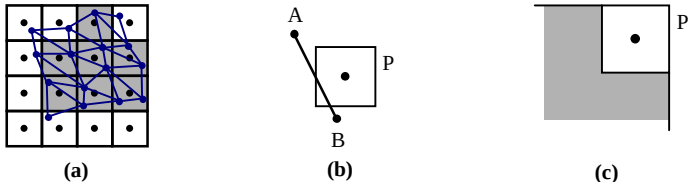


Figure 2: (a) Pixels contained in a mesh. **(b)** An impossible configuration. **(c)** A, B, C lie in the shaded region.

Corollary Suppose a surface is sampled at points which form a continuous triangular mesh on the surface. If the side length of every triangle in the mesh is less than the side length of a pixel at the target resolution (assuming unit magnification), then the surface is adequately sampled.

Proof This follows directly from the theorem and the fact that when the mesh is projected orthographically onto the screen, the projected side lengths of the triangles are less than or equal to their unprojected lengths.

2.2 Sampling

The previous section provides some direction as to how we can adequately sample an object. In particular, it suggests that to minimize the number of samples, the distance between adjacent samples on the surface of the object should be as large as possible but less than *ds*, the pixel side length at the target resolution (again assuming unit magnification). Now if we use a regular lattice to sample the orthographic views, which we consider a necessity for reasons of storage and rendering efficiency, then the distance between adjacent samples on the object could be arbitrarily large on surfaces which are nearly parallel to the viewing direction (figure 3a). However, suppose we restrict our attention to a surface (or partial surface) S whose normal differs by no more than θ from the projection direction at any point, where θ is a 'tolerance angle'. If we sample the orthographic view on an equilateral triangle lattice with side length $ds \cdot \cos\theta$, we obtain a continuous triangular mesh of samples on S in which each side length is at most $(ds \cdot \cos\theta) / \cos\theta = ds$ (figure 3b). Hence, S is adequately sampled.

If the object being modeled is smooth and convex, then all we need to do is choose a set of projections - which depend only on θ and not on the object itself - such that *every* part of the object's surface is adequately sampled by some projection. Unfortunately, most interesting objects exhibit a tendency to be neither smooth nor convex. We cannot, therefore, use the corollary of section 2.1 to 'prove' that an object is being adequately sampled. We simply use it to provide theoretical justification for an algorithm which, ultimately, must be verified experimentally.

We conclude with two comments on sampling. First, although we have referred multiple times to 'surface meshes' we do *not* store any surface mesh information with the point samples, nor is it necessary to do so. It suffices for there to exist *some* continuous triangulation of the points on a surface which satisfies the conditions given in section 2. Second, the preceding discussion brings to light the advantage of using an equilateral triangle mesh. If we were to use a square lattice instead, we would need to decrease the spacing between lattice points in order to compensate for the longer diagonals within the lattice squares, which would increase the total number of samples

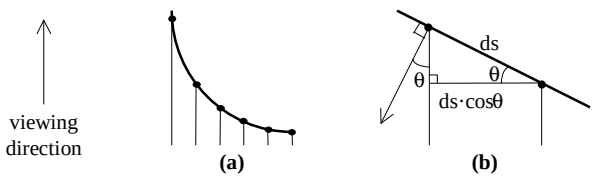


Figure 3: (a) Samples on the surface of the object can be arbitrarily far apart. **(b)** Using a tolerance angle to constrain the distance between adjacent samples.

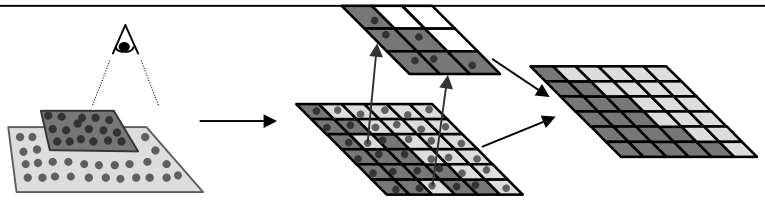


Figure 4: The dark surface is rendered using a higher level depth buffer in which there are no gaps. This depth buffer is used to detect and eliminate holes in the image.

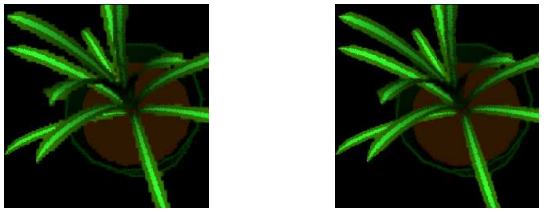
2.3 Magnification and Perspective

An adequately sampled object can be rendered orthographically without holes at the target resolution/magnification. To deal with magnification and perspective (which magnifies different parts of the object by different amounts), we introduce a *hierarchy* of lower resolution depth buffers as in [Green93]. The lowest buffer in the hierarchy has the same resolution as the target image, and each buffer in the hierarchy has one half the resolution of the buffer below it. Then, in addition to mapping point samples into the destination image as usual, each part of the object is rendered in a depth buffer at a low enough resolution such that the points cannot spread out and leave holes. When all the blocks have been rendered, we will have an image which, in general, will contain many gaps. However, it is now possible to detect and eliminate these holes by comparing the depths in the image to those in the hierarchy of depth buffers (figure 4). The next two sections provide details of this process.

2.4 Finding Gaps

Our strategy for detecting holes is to assign to each pixel a weight in [0, 1] indicating a 'confidence' in the pixel, where a 1 indicates that the pixel is definitely in the foreground and a 0 indicates that the pixel definitely represents a hole and its colour should be ignored. A simple way to assign these weights is to treat a pixel in the k^{th} depth buffer as an opaque square which covers exactly 4^k image pixels. We can then make a binary decision for each image pixel, assigning it a weight of either 0 or 1, by determining whether or not it lies behind some depth buffer pixel. This, however, produces pronounced blocky artifacts, as can be seen in Figure 5a.

A better approach is to instead treat the k^{th} depth buffer as a square mesh with vertices at the pixel centers. We then compute the amount by which the k^{th} depth buffer 'covers' an image pixel as follows: The center of the image pixel lies inside one of the squares of the mesh. The depth at each corner of the square is compared to



(a) **(b)**

Figure 5: Naïve weight assignment (left) results in blocky artifacts.

the depth of the image pixel; a corner is assigned a *coverage* of 1 if it lies in front of the pixel and 0 otherwise. Finally, we take the coverage of the pixel to be the bilinear interpolation of these four corner coverages. This is reminiscent of the "Percentage Closer Filtering" used in [Reeves87] to generate antialiased shadows.

The total coverage of a pixel is then the sum of the coverages from each depth buffer (capped at 1), and *weight* = 1 - *coverage.* In figure 5b we can see that this method produces much better results.

2.5 Filling Gaps

To fill the gaps in the final image we must compute a colour for pixels with weight less than 1. This problem is not specific to point sample rendering; it is the general problem of image reconstruction given incomplete information. Our solution is a variation of the two phase "pull-push" algorithm described in [Gortler96].

The 'pull' phase of the algorithm consists of computing a succession of lower resolution approximations of the image, each one having half the resolution of the previous one. We do this by repeatedly averaging 2x2 blocks of pixels, taking the weight of the new pixel to be the sum of the weights of the four pixels, capped at 1. In the 'push' phase, we use lower resolution images to fill gaps in the higher resolution images. For a pixel with colour c and weight $w < 1$, we compute an interpolated colour c_i using as interpolants the three closest lower resolution pixels with weights ½, ¼ and ¼ (figure 6a). The pixel's colour is then taken to be $wc + (1\text{-}w)c_i$.

(a) **(b)**

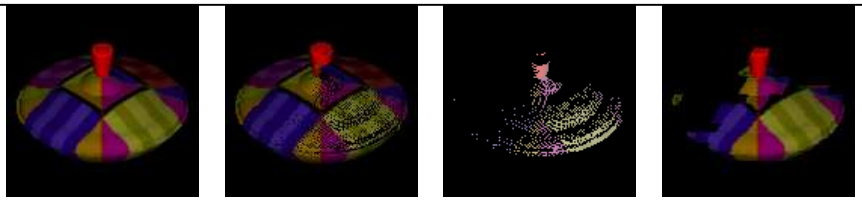
Figure 6: (a) An interpolated colour is computed using the three closest lower resolution pixels as shown. **(b)** Before gaps are filled (left) and after (right).

3 Object Modeling

Our software system is capable of automatically generating point sample representations for arbitrary synthetic objects; there is no user intervention. A target resolution is specified, and the object is scaled to fit inside a square window having this resolution. This implicitly sets the target magnification.

A fixed set (i.e. independent of the object being modeled) of 32 orthographic projections are sampled on an equilateral triangle lattice. The density of this lattice is determined by the tolerance angle described in section 2.2; we used 25° for all models in this paper. The samples in a projection are divided into 8x8 *blocks* which allows us to compress the database by retaining only those blocks which are needed, while maintaining the rendering speed and storage efficiency afforded by a regular lattice.

We use a greedy algorithm to find a set S of blocks which provides an adequate sampling of the object with as little redundancy as possible. We start with no blocks



(a) **(b)** **(c)** **(d)**

Figure 7: Block Selection. **(a)** Ray traced projection. **(b)** Projection reconstructed from current set of blocks. **(c)** Pixels where (a) and (b) have different depths. **(d)** Blocks added to set.

($S = \phi$). We then step through the list of orthographic projections; from each one we select blocks to add to S. A block is selected if it corresponds to a part of the object which is not yet adequately sampled by S. Rather than attempt to exactly determine which surfaces are adequately sampled, which is very difficult, we employ the heuristic of both ray tracing the orthographic projection and reconstructing it from S, then comparing the depths of the two images thus obtained (figure 7). This tests for undersampled surfaces by searching for pixels which are 'missed' by the reconstruction.

4 Rendering

There are five steps in the point sample rendering pipeline:

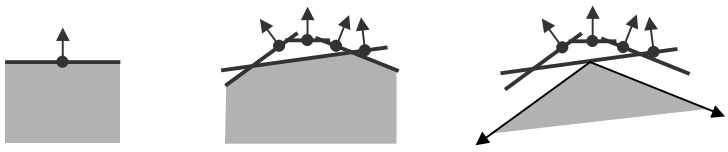
Visibility Testing We conservatively estimate a set of visible blocks using the view frustum and two additional data structures described in section 4.1.
Block Warping Each potentially visible block is 'warped' [Wolberg90] into the destination image. This is described in section 4.2.
Finding Gaps We assign weights to the image pixels as described in section 2.4
Shade Each non-background pixel is shaded according to the current lighting model.
Filling Gaps Pixels with weight less that one are coloured as described in section 2.5.

4.1 Visibility Data Structures

In addition to the view frustum, we make use of two per-block data structures to speed up rendering by conservatively estimating a set of visible blocks.

4.1.1 Visibility Cones

The tangent plane at each point sample defines a half space from which the point is never visible. Taking the intersection, over all the points in a block, of these half spaces, we obtain a region in space, bounded by up to 64 planes, from which none of



(a) **(b)** **(c)**

Figure 8: **(a)** Half space from which point is not visible **(b)** Region from which block is not visible **(c)** Cone inside region.

the block is ever visible. We place a cone entirely inside this region. This 'visibility cone' gives us a fast, conservative visibility test; none of the block is visible from any viewpoint within the cone (figure 8).

4.1.2 Visibility Masks

The visibility mask is based on Normal Masks as introduced by Zhang and Hoff [Zhang97]. The sphere of directions is subdivided into 128 spherical triangles. For each block we create a bitmask of length 128 - one bit per triangle. The k^{th} bit in a block's bitmask is 1 if and only if the block is visible (from a viewpoint outside the convex hull of the object) from some direction which lies inside the k^{th} triangle. We call this bitmask the *visibility mask* for that block.

The simplest way to make use of these masks at render time is to construct a single visibility mask for the entire screen. In this mask, the k^{th} bit is set if and only if some direction from the eye to a screen pixel lies inside the k^{th} triangle on the triangulated sphere of directions. We then AND this mask with each block's mask; a block is definitely not visible if the result of this test is zero.

As explained in [Zhang97], however, this is overly conservative. We are, in effect, using the entire set of directions from the eye to the screen to approximate the set of directions subtended by a single block. In reality the latter set is almost certainly much smaller, so if we can improve the approximation then we will improve the rate of culling. To this end we subdivide the screen into 2^n x 2^n square regions. We compute a mask for each region, and we use a BSP tree to quickly determine which region a block lies in, and therefore which mask to use to test the block's visibility.

There is naturally a tradeoff involved in choosing n - larger n will result in improved culling, but more overhead. We have found that n = 2 gives the best performance on average (i.e. a 4 x 4 subdivision of the screen).

4.2 Block Warping

The points in each potentially visible block must be transformed to screen space and mapped to image pixels. This procedure uses incremental calculations and is very fast. For each block, we also select a depth buffer with a low enough resolution such that block's points cannot spread out and leave holes. Rather than storing the points' exact depths in this buffer, we first subtract a small threshold. The reason for this is that the depth buffer will be used to filter out points which lie behind the foreground

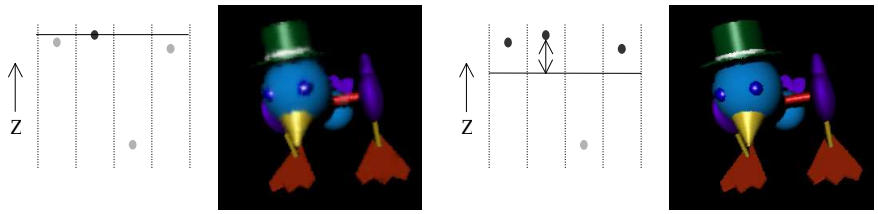


Figure 10: Four points are mapped to different image pixels contained in a single depth map pixel. **(a)** No depth threshold - only the nearest point is retained. **(b)** Corresponding image. **(c)** The use of a small threshold prevents points on the same surface as the nearest point from being discarded. **(d)** Corresponding image.

surface; the use of a threshold prevents points which lie *on* the foreground surface from being accidentally discarded. Figure 10 illustrates this in one dimension and shows the striking difference in image quality which results from the use of a threshold. We have found a good threshold to be three times the image pixel width.

4.3 Shading

In our test system we have implemented Phong shading with shadows. As shading is performed in screen space after all blocks have been warped, points' normals and shininess must be copied into the image buffer in addition to their colours.

There are two points worth noting here. First, since the specular component of illumination is computed at each pixel, the resulting highlights are of ray-trace quality (figure 11a). Second, since the creation of the shadow maps [Williams78] used to compute shadows involves only fast block warping without the overhead of copying normals, they can be constructed very quickly. For example, figure 11b was rendered at 256x256 with three light sources. It took 204ms to render without shadows and 356ms with shadows - an overhead of less than 25% per light source

(a) **(b)**

Figure 11: (a) High quality specular highlights **(b)** fast shadows

5 Results

Our test system was run as a single process on a 333MHz dual Pentium II with 512MB RAM. Table 1 gives file size, number of blocks and construction time for the four models shown in the appendix. The models were rendered with no shadows and a single directional light at 256x256 pixels; table 2 gives timing statistics for these settings. The largest of the models, 'pine tree', was obtained from an Inventor model which contains 15,018 cones and ellipsoids and takes over 5 seconds to render using the Inventor Scene Viewer running on the same machine.

Table 3 shows the extent to which the visibility cone and visibility mask are able to eliminate blocks. Note that the visibility cone works quite well for objects with low curvature (e.g. Top) but poorly for objects with high curvature (e.g. Chair, Pine Tree). This is due to the fact that surfaces with high curvature are visible from a larger set of directions. In all cases the visibility cone is outperformed by the visibility mask, but it is still able to provide some additional culling.

Table 1: Model statistics

	Chair	**Pine Tree**	**Sailboat**	**Top**
file size (MB)	1.32	7.12	1.00	2.96
number of blocks	2666	11051	1786	4473
construction time (hrs : min)	0:15	1:35	0:13	0:43

Table 2: Timing for a software only implementation on a 333MHz Pentium II.

time (ms)	Chair	Pine Tree	Sailboat	Top
initialize buffers & screen masks	9	9	9	9
visibility (cone and mask)	3	9	2	6
warp blocks	44	226	33	61
find gaps	3	4	6	3
shade	14	25	18	21
fill gaps	9	9	9	8
total	**82**	**282**	**77**	**108**

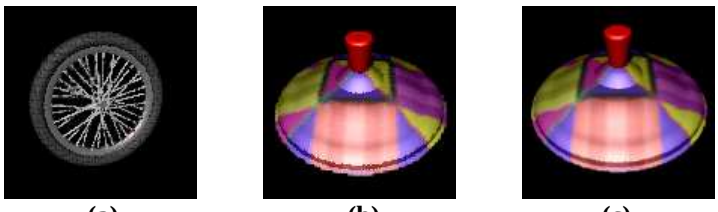
Table 3: % blocks eliminated as a function of which visibility tests are used.

% culled	Chair	Pine Tree	Sailboat	Top
neither	0	0	0	0
cone only	13.9	1.1	24.9	42.9
mask only	23.1	9.5	33.4	43.3
both	26.1	9.9	35.8	51.5

6 Discussion and Future Work

The key to the speed of our algorithm is the simplicity of the procedure used to warp points into the destination image. All of the relatively expensive computations - shading, finding gaps, and filling gaps - are performed in screen space after all points have been warped. This places an absolute upper bound on the overhead of these operations, independent of the number and/or complexity of models being rendered. Ideally, one would also like the overall rendering time to be independent of model complexity. Although this is not the case, it is very nearly so; for example, the 'pine tree' inventor model is two orders of magnitude more complex than 'sailboat', yet the point sample rendering time is less than four times greater.

One limitation of the algorithm as described is that it is difficult to properly sample thin spoke-like structures which are on the order of one pixel wide. As a result, such structures often appear ragged in rendered images (figure 12). One possible solution to this problem, which is itself an interesting area of investigation, is to have objects generate their own point samples rather than relying on some external sampling process. Each geometric primitive would produce a set of points which form an adequate sampling of the primitive, much in the same way that primitives 'dice' themselves into micropolygons in the Reyes rendering architecture [Cook87]. The difficulty lies in organizing these generated points into blocks or some other structure so that they may be stored and rendered efficiently.



(a) **(b)** **(c)**

Figure 12: (a) Ragged spokes **(b)** Rough edges **(c)** Rendered at 256x256 and displayed at 128x128 (2x2 subpixel anti-aliasing).

Another weakness of point sample rendering is that without the precise geometric information available in polygon rendering, it is difficult to perform any sort of sub-pixel anti-aliasing, particularly at silhouette edges. A brute force solution is to increase the sampling density and render objects at double the image resolution. This works well (figure 12), but has the obvious disadvantage of being significantly slower. A more sophisticated variant of this solution would be to store both low and high resolution copies of each block, using the slower high resolution copy only for those blocks that lie on a silhouette edge.

A topic which invariably arises in any discussion of modeling paradigms is the acquisition of models from real objects. In order to construct a point sample model for an object, it is necessary to somehow obtain both shape and reflectance information for that object. This is a challenging problem, but it has been addressed quite successfully by Sato, Wheeler and Ikeuchi [Sato97a]. In fact, one of the problems they encountered was that the models they generated - which included dense grids containing colour, normal and specular information - could not be rendered using standard software or hardware. It was therefore necessary for them to write software to display these models, and image generation was slow [Sato97b]. This problem would be solved by generating point sample representations of these models which would make full use of the reflectance information and could be rendered in real time.

Acknowledgments

We would like to thank the following people for granting us permission to use their Inventor models: Barbara Cutler (‘sailboat’), Brendon Glazer (‘toybird’), Michael Golding (‘gavel’), Stephen Ho (‘tree’ and ‘pine tree’), and Tara Schenkel (‘chair’). These models were created as part of a computer graphics course taught by Seth Teller at M.I.T. We would also like to thank Seth Teller, Pat Hanrahan, John Owens, Scott Rixner and Shana Nichols for their valuable comments. This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under DARPA Order E254 and monitored by the Army under Contract DABT63-96-C-0037.

References

[Chen93] Shenchang Eric Chen, Lance Williams, “View Interpolation for Image Synthesis”, SIGGRAPH ’93 Proceedings, pp. 279-285

[Chen95] Shenchang Eric Chen, “QuickTime® - An Image-Based Approach to Virtual Environment Navigation”, SIGGRAPH ’95 Proceedings, pp. 29-37

[Cook87] Robert L. Cook, Loren Carpenter, Edwin Catmull, “The Reyes Image Rendering Architecture”, SIGGRAPH ’87 Proceedings, pp. 95-102

[Csuri79] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, M. Howard, “Towards an Interactive High Visual Complexity Animation System”, SIGGRAPH ’79 Proceedings, pp. 289-299

[Curless96] Brian Curless, Marc Levoy, “A Volumetric Method for Building Complex Models from Range Images”, SIGGRAPH ’96 Proceedings, pp. 303-312

[Dally96] William J. Dally, Leonard McMillan, Gary Bishop, Henry Fuchs, “The Delta Tree: An Object-Centered Approach to Image-Based Rendering”, Artificial Intelligence memo 1604, Massachusetts Institute of Technology, May 1996.

[Gortler96] Stephen J. Gortler, Radek Grzeszczuk, Richard Szeliski, Michael F. Cohen, “The Lumigraph”, SIGGRAPH ’96 Proceedings, pp. 43-54

[Greene93] Ned Greene, Michael Kass, Gavin Miller, "Hierarchical Z-buffer Visibility", Proc. SIGGRAPH '93 Proceedings, pp. 231-238

[Heckbert87] Paul S. Heckbert, "Ray Tracing Jell-O® Brand Gelatin", *Computer Graphics* (SIGGRAPH '87 Proceedings), Vol. 21, No. 4, July 1987, pp. 73-74

[Hoppe92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, Werner Stuetzle, "Surface Reconstruction from Unorganized Points", SIGGRAPH '92 Proceedings, pp. 71-78

[Laveau94] S. Laveau, O.D. Faugeras, "3-D Scene Representation as a Collection of Images and Fundamental Matrices", INRIA Technical Report No. 2205, February 1994

[Levoy85] Mark Levoy, Turner Whitted, "The Use of Points as a Display Primitive", Technical Report TR 85-022, The University of North Carolina at Chapel Hill, Department of Computer Science, 1985

[Levoy96] Mark Levoy, Pat Hanrahan, "Light Field Rendering", SIGGRAPH '96 Proceedings, ACM SIGGRAPH, pp. 31-42

[Mark97] William R. Mark, Leonard McMillan, Gary Bishop, "Post-Rendering 3D Warping", Proc. 1997 Symposium on Interactive 3D Graphics, pp. 7-16

[Max95] Nelson Max, Keiichi Ohsaki, "Rendering Trees from Precomputed Z-Buffer Views", 6th Eurographics Workshop on Rendering, June 1995, pp. 45-54

[McMillan95] Leonard McMillan, Gary Bishop, "Plenoptic Modeling: An Image-Based Rendering System", SIGGRAPH '95 Proceedings, pp. 39-46

[Pulli97] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, Werner Stuetzle, "View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data", *Rendering Techniques '97*, Proc. Eurographics Workshop, pp. 23-34

[Reeves83] William T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", SIGGRAPH '83 Proceedings, pp. 359-376

[Reeves85] W.T. Reeves, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", SIGGRAPH '85 Proceedings, pp. 313-322

[Reeves87] William T. Reeves David H. Salesin, Robert L. Cook, "Rendering Antialiased Shadows with Depth Maps", SIGGRAPH '87 Proceedings, pp. 283-291

[Smith84] Alvy Ray Smith, "Plants, Fractals and Formal Languages", *Computer Graphics* (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 1-10

[Sato97a] Yoichi Sato, Mark D. Wheeler, Katsushi Ikeuchi, "Object Shape and Reflectance Modeling from Observation", SIGGRAPH '97 Proceedings, pp. 379-387

[Sato97b] Yoichi Sato, Personal Communication (Question period following presentation of [Sato97a], SIGGRAPH 1997)

[Schaufler97] Gernot Schaufler, "Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes", 8th Eurographics Workshop on Rendering, pp. 151-162

[Turk94] Greg Turk and Marc Levoy, "Zippered Polygon Meshes from Range Images", Proc. SIGGRAPH '94 Proceedings, pp. 311-318

[Westover90] Lee Westover, "Footprint Evaluation for Volume Rendering", *Computer Graphics* (SIGGRAPH '90 Proceedings), Vol. 24, No. 4, 1990, pp. 367-376.

[Williams78] Lance Williams, "Casting Curved Shadows on Curved Surfaces", SIGGRAPH '78 Proceedings, pp. 270-274

[Wolberg90] G. Wolberg, **Digital Image Warping,** IEEE Computer Society Press, Los Alamitos, California, 1990.

[Wong97] Tien-Tsin Wong, Pheng-Ann Heng, Siu-Hang Or, Wai-Yin Ng, "Image-based Rendering with Controllable Illumination", *Rendering Techniques '97*, Proc. Eurographics Workshop, pp. 13-22

[Zhang97] Hansong Zhang, Kenneth E. Hoff III, "Fast Backface Culling Using Normal Masks", Proc. 1997 Symposium on Interactive 3D Graphics, pp. 103-106