The early history of point-based graphics

Marc Levoy
Stanford University

Why is it worthwhile to study where an idea came from? Thomas Kuhn, writing in *The Structure of Scientific Revolutions*, notes that scientists like to see their discipline's past "developing linearly toward its present vantage." As a result, textbooks often discard or obscure the origins of ideas, thereby robbing students of the experience of a scientific revolution. This in turn makes them unable to realize when one is upon them, and ignorant about how to act in these circumstances. I do not claim that point-based rendering was a scientific revolution, at least not in 1985 when Turner Whitted and I wrote our first paper on the topic. However, that paper was written in response to a scientific *crisis*, which bears some of the same characteristics. As a technical achievement, our paper was a failure. However, as a story of crisis and response it is instructive. In this spirit I offer the following historical account.

Sample-based representations of geometry

Since the beginning of computer graphics, a creative tension has existed between representing scenes as geometry versus as collections of samples. Early sample-based representations included textures, sprites, range images, and density volumes. More recent examples include light fields, layered depth images, image caches, and so on. Points are another such representation, often used to approximate curved surfaces as this book amply demonstrates. In each case researchers faced a common set of challenges: how to edit the scene by manipulating its samples, how to store and compress these samples, how to transform and shade them, and how to render them with correct sampling, visibility, and filtering.

However, to understand the early history of point rendering, we must understand a different tension that existed in the early history of computer graphics - between image-order and object-order algorithms for displaying geometric primitives. It was in response to this tension that Turner Whitted and I proposed points as a way to display curved surfaces [1985]. And it was on the shoals of sampling, visibility, and filtering that our idea ran aground. Let us see why.

Image-order versus object-order visibility and antialiasing

In his seminal paper on hidden-surface algorithms, Ivan Sutherland [1974] showed that visibility is tantamount to sorting. As any student of computing knows, sorting N objects into P bins can be done using a gather or a scatter. In computer graphics, the gather strategy leads to an image-order algorithm. One example is ray tracing [Whitted 1980]; for the viewing ray associated with each image pixel, search among the geometric primitives in a scene for the frontmost primitive intersecting that ray. By contrast, the scatter strategy leads to an object-order algorithm. The most common of these is the Z-buffer [Catmull 1974]; create an array as large as the screen, and for each primitive decide which pixel it falls into. If the primitive covers many pixels, one can traverse it in image order, for example using a scanline algorithm [Watkins 1970]. Image-order traversal is particularly easy to implement because the number of samples that should be taken of the primitive is obvious: one per pixel. For an object-order algorithm, enough samples must be taken to avoid leaving any pixels uncovered, but not so many that the algorithm becomes inefficient.

To avoid aliasing artifacts in computer-generated images, each pixel should be assigned not a point sample of the scene but instead a sample of the convolution of the scene by a filter function. Repeating this process for every pixel in a 2D image, and assuming the filter is a discrete 2D function, we obtain four nested loops. Since convolution is linear, these loops can be rearranged so that the outer loop is over image pixels, leading to an image-order algorithm, or over points on the scene primitives at some resolution, leading to an object-order algorithm. As was the case for visibility, antialiasing poses fewer problems if implemented in image order. In an influential early paper, Edwin Catmull [1978] observed that to compute a correct color in a pixel, only those primitives or portions of primitives that lie frontmost within the filter kernel centered at the pixel should be included in the convolution. This is easy in an image-order algorithm, because all primitives that might contribute to the pixel are evaluated at once. In an object-order algorithm, solving this problem requires retaining subpixel geometry for every primitive in every pixel. To avoid this difficulty, researchers have proposed computing visibility at a higher resolution than the pixel spacing (by supersampling and averaging down), approximating subpixel geometry using a bitmask [Carpenter 1984], or summarizing it as a scalar value (called *alpha*), leading to digital compositing [Wallace 1981, Porter 1984]. In some rendering algorithms, subpixel geometry has been used as both an alpha value and a filter weight, leading to problems of correctness to which I will return later.

The challenge posed by procedural modeling

If it is easier to render scenes in image-order, why did researchers develop object-order algorithms? The answer lies in the convenience of procedural modeling, which may be loosely defined as the generation of scene geometry using a computer algorithm (rather than interactively or by sensing). Examples of procedural modeling include fractal landscapes [Carpenter 1980], clouds [Gardner 1985], plants [Prusinkiewicz 1990], and generative surface models [Snyder 1992]. Although some cite Levoy and Whitted [1985] as introducing points as primitives, procedurally-generated points or particles had already been used to model smoke [Csuri 1979], clouds [Blinn 1982], fire [Reeves 1983], and tree leaves and grass [Reeves 1985].

To render a procedurally defined object using favored image-order algorithms, one must be able to compute for a given pixel which part of the object (if any) lands there. If the procedure is expensive to invert in this sense, or even uninvertible, then an object-order algorithm rendering must be used. During the 1970s and early 1980s, researchers invested considerable effort in resolving this conflict - between rendering order and geometry traversal order. As an example, Reeves [1985] modeled tree leaves as circular particles with semi-transparent fringes. To decide how many particles to draw for each tree, he examined its approximate size on the screen. He rendered these particles using an image-order algorithm. In this algorithm, transparency could be used either as a filter weight or a compositing alpha, but not both, as noted earlier. To resolve this ambiguity, Reeves sorted his particles into buckets by screen location and Z-depth, treated transparency as weight, and additively accumulated color and weight in each pixel. When a bucket was finished, it would be combined with other buckets using digital compositing, with the accumulated weight in each pixel now serving as its alpha value. While not exact, this algorithm worked well for irregular geometry like trees and grass.

Another important class of procedurally defined objects are parametric surfaces. For given values of the parameters s and t, it is straightforward to evaluate the surface functional, yielding an (x, y) position on the screen. However, for a given pixel position it may be difficult to determine whether the surface touches it. For parametric bicubic surfaces, some researchers attacked this inverse problem head-on, developing scanline algorithms that directly gave these curves of intersection [Blinn 1978, Whitted 1978]. However, these algorithms were fragile and difficult to implement efficiently. Others proposed an object-order approach, subdividing the surface recursively in parametric space into patches until their projection covered no more than one pixel [Catmull 1974]. Still others proposed hybrid solutions, subdividing the surface recursively until it was locally flat enough [Clark 1979, Lane 1980] (or detailed enough in the case of fractal surfaces [Carpenter 1980]) to represent using a simpler primitive, which could be rendered using an image-order algorithm. Another hybrid solution was to partially evaluate the procedural geometry, producing an estimate of its spatial extent in the form of an image space decomposition [Rubin 1980] or collection of bounding boxes [Kajiya 1983]; the overlap between these extents and screen pixels could then be evaluated in image order.

This struggle, which had to be repeated each time a new geometric primitive was proposed, was perceived by many as a crisis in the field. Kuhn [1962] states that when such a crisis arises in a scientific paradigm, and "normal science" is no longer fruitful, there is a gradual loosening of the rules for research, leading researchers to propose solutions that were previously considered outlandish. Before we make that jump, there is one procedural modeling method that so severely broke the dominant display paradigm that it requires special mention.

The curious case of displacement mapping

One of the most important single advances in the realism of computer imagery was texture mapping, first demonstrated by Catmull [1974]. By associating a tabular two-dimensional array of colors with each geometric primitive, the visual complexity of a scene could far exceed its shape complexity. Bump mapping [Blinn 1978] generalized texture mapping by using a tabular array to locally modify surface orientation.

In his paper on shade trees, Cook proposed using textures to locally modify surface position [1984]. Although a powerful idea, displacement mapping badly broke the accepted image-order graphics pipeline. In this pipeline the set of pixels covered by a primitive was independent of whatever textures were associated with it; a primitive could be rasterized without regard to its textures, and once a pixel was selected by the rasterizer for rendering, its textures could be considered when computing its color. In Cook's method a texture was allowed to move a primitive and thereby change the set of pixels it covered.

For the restricted case of a flat plane displaced in one direction only (i.e. a terrain model), this problem could be fixed by changing the screen traversal order, as described by Fishman [1980] and Max [1981]. However, if the displacement could be in any direction, or the underlying geometric primitive was curved, then the problem became harder. Moreover, since displacement maps could be generated procedurally, interactively, or by sensing, their displacement function was often uninvertible. This made the problem insoluable except by abandoning image-order rendering.

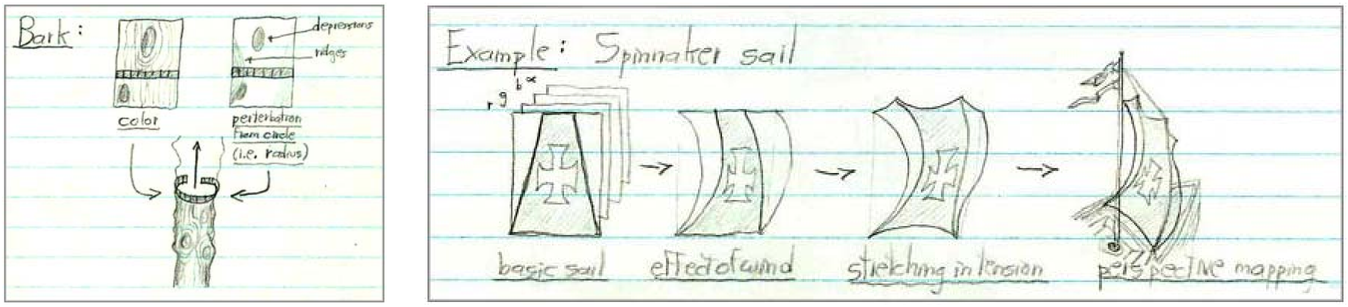

Figure 1: Levoy and Whitted envisioned modeling complex objects from points, represented as textures and compounded using simple rules. At left, a tree trunk is modeled using two textures - giving color and X,Y,Z-displacement from the surface of a cylinder. At right, a trapezoidal texture is warped using a sequence of mappings - each defined as a texture or procedure. If the compound mapping is uninvertible, then rendering can only be done in object order. These sketches are from Marc Levoy's research notebook of September 1984.

Points and micropolygons to the rescue

The notion of using points and an object-order rendering algorithm to display smooth (i.e. continuous) primitives was first proposed for lines and curved strokes by Whitted [1983] and for displacement-mapped surfaces by Levoy and Whitted [1985]. Although the latter paper advertised points as a universal meta-primitive, having in mind the rather ambitious modeling paradigm described in figure 1, they only demonstrated its use for rendering displaced surfaces.

The notion of using points to display continuous surfaces was not an obvious idea when proposed, nor was it obviously a good idea once proposed. Indeed, the method immediately ran into exactly the problems enumerated at the beginning of this chapter. The first of these was how to vary the density of points locally to match the pixel spacing. If this density was too high, the algorithm would be slow; if it was too low, the surface would contain holes. To solve this problem, Levoy and Whitted used the determinant of the Jacobian of a unit-sized surface patch after transformation to screen space, an idea that resurfaced in Heckbert's analysis of texture mapping [1989].

Harder to resolve was the conflict between visibility and antialiasing. For this Levoy and Whitted proposed (concurrently but independently from Reeves) a "moving surface cache", composed of depth buckets that were dynamically created and destroyed as the surface was traversed. Like Reeves, points were rendered with antialiasing, whose weights were combined additively within a bucket and multiplicatively (via digital compositing) between buckets. Unfortunately, while Reeves got away with this approximation because he was rendering irregular objects, Levoy and Whitted did not. Their use of a moving cache was clever, but it was also fragile; narrow surfaces, highly curved surfaces, and surfaces that folded over on themselves sometimes exhibited holes, almost regardless of the density of points employed.

The REYES algorithm [Cook 1987] solved this same problem - of rendering displacement-mapped surfaces - by traversing them in object order, dicing each into "micropolygons" about the size of a pixel, and rendering these using an image-order algorithm with stochastic sampling [Cook 1986]. Although arguably more expensive than point rendering, Cook's micropolygons fit snugly against each other. An image-order algorithm could render such a mesh of micropolygons without creating holes. In retrospect, Levoy and Whitted's mistake was to discard the natural connectivity between points on a surface. In so doing, they made reconstruction of a continuous surface difficult.

Not to belabor our story, the publication and obvious success of the REYES algorithm caused Levoy and Whitted to abandon their work on point-based rendering. Points might still be advantageous, they reasoned, but their advantages must lay elsewhere. Indeed, researchers continued throughout the 1980s experimenting with point-like approaches in other domains. For example, under the name "splatting" [Westover 1990], antialiased points briefly became one of the two dominant methods for rendering volume data, the other two being ray tracing [Levoy 1988] and planar texture mapping [Drebin 1988]. For the special case of displaying isosurfaces from volume data, a point-like algorithm called "dividing cubes" was proposed by Cline and Lorensen [1987], whereby a volume was subdivided into subvolumes until their projection covered no more than a pixel. Although implemented in hardware, this algorithm was eventually overshadowed in popularity by the same authors' Marching Cubes algorithm [Lorensen 1987].

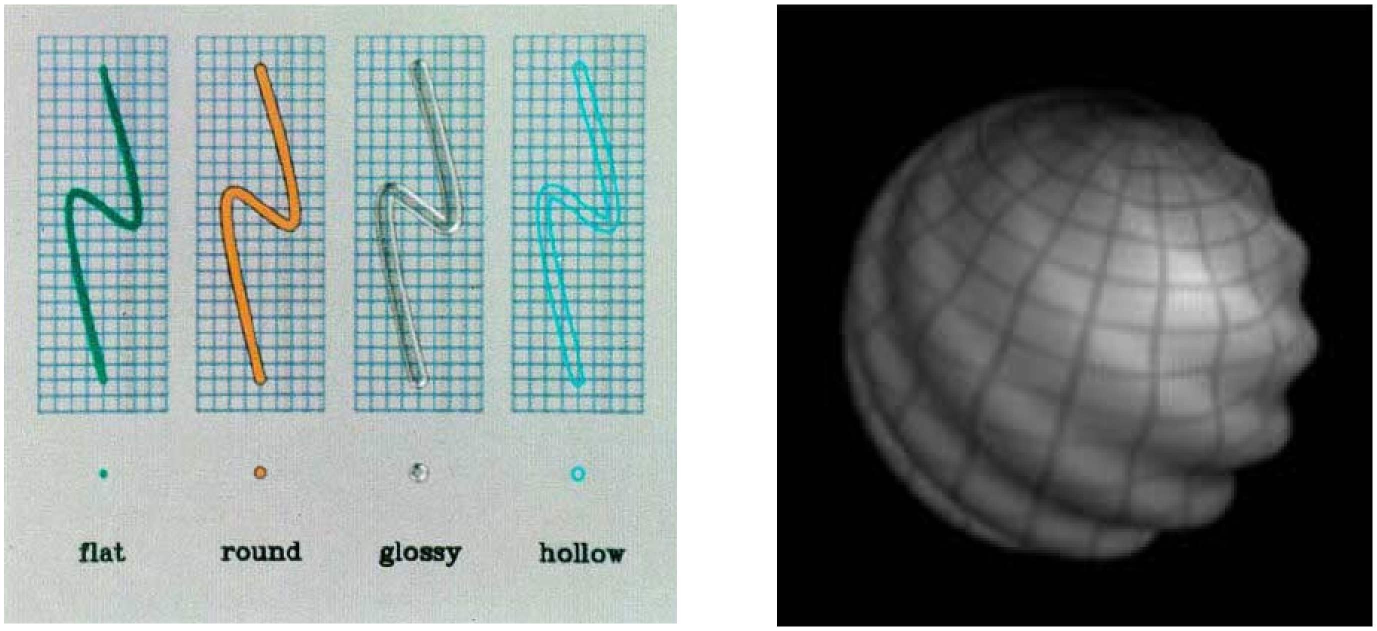

Figure 2: At left, Whitted [1983] rendered continuous curves as a sequence of brush stamps in 3D, each represented as a pixel array having with color, opacity, and Z-depth. At right, Levoy and Whitted [1985] rendered displacement-mapped surfaces (i.e. surfaces with relief) as a collection of points in 3D, each represented as a pixel array having the same characteristics.

The current Renaissance in point-based graphics

Kuhn says that crises in science end in one of three ways: (1) the existing paradigm proves capable of handling the crisis, (2) the problem is set aside for future generations having better tools, or (3) there is a transition to a new paradigm. Although I have argued here that the crisis in rendering algorithms in the 1980s was adequately resolved without a switch to point-based rendering, this is not to say that all the difficulties posed by the dominant paradigm were resolved. Micropolygon rendering is slow and parallelizes poorly, and the out-of-order evaluation demanded by displacement maps still causes headaches to the designers of graphics hardware. Nevertheless, except for one paper in the mid-1990's on a hardware system for point-based rendering [Grossman 1998], the idea was more or less forgotten for a decade.

By the time point-based rendering was resurrected (contemporaneously by Rusinkiewicz and Levoy in their QSplat system [2000] and by Pfister et al. in their Surfels system [2000]), the graphics landscape had changed considerably. The number of pixels covered by a typical polygon had been shrinking for a decade, and hardware antialiasing was becoming commonplace. Both developments made antialiased points a more attractive primitive. At the same time, display screen resolution was rising slowly, which made accurate antialiasing less critical than it was 15 years earlier. In addition, the 1990s saw the development of several new ways to create points, including 3D scanning and particle-based physics simulations. In some cases these point sets included connectivity information, but in other cases they did not, leading to so-called *point cloud* data. Finally, new techniques had been invented for discretely approximating the operators of differential geometry, enriching the set of operations that could be applied to point-based representations of surfaces.

With the lesson of the rise and fall of point-based rendering in the 1980s in mind, and in my unenvious position as the author of this rise and fall, I encourage the new generation of researchers in this field, many of whom are represented in this book, to learn from my mistake - by clearly distinguishing those things points can do, but so can other representations, those things points can do that other representations cannot, and those things points cannot do or will never do well.

References

[Blinn 1978] Blinn, J.F., *Computer display of curved surfaces*, PhD thesis, Department of Computer Science, University of Utah, 1978.

[Blinn 1982] Blinn, J.F., "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces," *Proc. SIGGRAPH 1982*.

[Carpenter 1980] Carpenter, L., "Rendering of Fractal Curves and Surfaces," *Proc. SIGGRAPH 1980*.

[Carpenter 1984] Carpenter, L., "The A-buffer, an antialiased hidden surface method," *Proc. SIGGRAPH 1984*.

[Catmull 1974] Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD thesis, Department of Computer Science, University of Utah, December 1974.

[Catmull 1978] Catmull, E., "A hidden-surface algorithm with anti-aliasing," *Proc. SIGGRAPH 1978*.

[Clark 1979] Clark, J., "A fast scan-line algorithm for rendering parametric surfaces," *Proc. SIGGRAPH 1979*.

[Cline 1988] Cline, H.E., Lorensen, W.E., Ludke, S., Crawford, C.R., and Teeter, B.C., "Two Algorithms for the Three-Dimensional Reconstruction of Tomograms," *Medical Physics*, Vol. 15, No. 3, May/June, 1988.

[Cook 1984] Cook, R., "Shade Trees," *Proc. SIGGRAPH 1984*.

[Cook 1986] Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics,* Vol. 5, No. 1, January, 1986.

[Cook 1987] Cook, R.L., Carpenter, L., Catmull, E., "The Reyes image rendering architecture," *Proc. SIGGRAPH 1987*.

[Csuri 1979] Csuri, C., Hackathorn, R., Parent, R., Carlson, W. and Howard, M., "Towards an Interactive High Visual Complexity Animation System," *Proc. SIGGRAPH 1979*.

[Drebin 1988] Drebin, R.A., Carpenter, L., and Hanrahan, P., "Volume Rendering," *Proc. SIGGRAPH 1988*.

[Fishman 1980] Fishman, B., Schachter, B., "Computer Display of Height Fields," *Proc. SIGGRAPH 1980*.

[Gardner 1985] Gardner, G., "Visual Simulation of Clouds," *Proc. SIGGRAPH 1985*.

[Grossman 1998] Grossman, J.P., Dally, W.J., "Point Sample Rendering," *Proc. Eurographics Workshop on Rendering Techniques*, 1998.

[Heckbert 1989] Heckbert, P., *Fundamentals of Texture Mapping and Image Warping*, Master's thesis, U.C. Berkeley, June 1989.

[Kajiya 1983] Kajiya, J.T., "New Techniques for Ray Tracing Procedurally Defined Objects," *Proc. SIGGRAPH 1983*.

[Kuhn 1962] Kuhn, T., *The Structure of Scientific Revolutions*, University of Chicago Press, 1962.

[Lane 1980] Lane, J.C., Carpenter, L., Whitted, T., Blinn, J., "Scan line methods for displaying parametrically defined surfaces," *CACM*, Vol. 23, No. 1, 1980.

[Levoy 1985] Levoy, M. and Whitted, T., "The Use of Points as a Display Primitive," Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, January, 1985.

[Levoy 1988] Levoy, M., "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications,* Vol. 8, No. 3, May, 1988.

[Lorensen 1987] Lorensen, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Proc. SIGGRAPH 1987*.

[Max 1981] Max, N.L., "Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sunset," *Proc. SIGGRAPH 1981*.

[Pfister 2000] Pfister, H., Zwicker, M., Baar, J.v., Gross, M., "Surfels: Surface Elements as Rendering Primitives," *Proc. SIGGRAPH 2000*.

[Porter 1984] Porter, T., Duff, T., "Compositing Digital Images," *Proc. SIGGRAPH 1984*.

[Prusinkiewicz 1990] Prusinkiewicz, P., Lindenmayer, A., *The algorithmic beauty of plants*, Springer-Verlag, 1990.

[Reeves 1983] Reeves, W.T., "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects," *Proc. SIGGRAPH 1983*.

[Reeves 1985] Reeves, W.T., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," *Proc. SIGGRAPH 1985*.

[Rubin 1980] Rubin, S.M. and Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Proc. SIGGRAPH 1980*.

[Rusinkiewicz 2000] Rusinkiewicz, S., Levoy, M., "Qsplat: a multiresolution point rendering system for large meshes," *Proc. SIGGRAPH 2000*.

[Snyder 1992] Snyder, J.M., *Generative Modeling for Computer Graphics and CAD*, Academic Press, 1992.

[Sutherland 1974] Sutherland, I.E., Sproull, R.F., Schumacker, R.A., "A Characterization of Ten Hidden-Surface Algorithms," *ACM Computing Surveys*, Vol. 6, No. 1, 1974.

[Watkins 1970] Watkins, G.S., *A real-time visible surface algorithm*, Technical Report UTECH-CSc-70-101, University of Utah, 1970.

[Wallace 1981] Wallace, B., "Merging and Transformation of Raster Images for Cartoon Animation," *Proc. SIGGRAPH 1981*.

[Westover 1990] Westover, L., "Footprint Evaluation for Volume Rendering", *Proc. SIGGRAPH 1990*.

[Whitted 1978] Whitted, T., "A scan line algorithm for computer display," *Proc. SIGGRAPH 1978*.

[Whitted 1980] Whitted, T., "An improved illumination model for shaded display," *CACM*, Vol. 23, No. 6, June 1980.

[Whitted 1983] Whitted, T., "Anti-aliased line drawing using brush extrusion," *Proc. SIGGRAPH 1983*.