Image Rendering by Adaptive Refinement

Larry Bergman, Henry Fuchs, Eric Grant
University of North Carolina at Chapel Hill

Susan Spach
Hewlett-Packard Laboratories
Palo Alto, California

Abstract

This paper describes techniques for improving the performance of image rendering on personal workstations by using CPU cycles going idle while the user is examining a static image on the screen. In that spirit, we believe that *a renderer's work is never done*. Our goal is to convey the most information to the user as early as possible, with image quality constantly improving with time. We do this by first generating a crude image rapidly and then adaptively refining it where necessary as long as the user does not change viewing parameters. The renderer operates in a succession of phases, first displaying only vertices of polygons, next polygon edges, then flat shading polygons, then shadowing polygons, then Gouraud shading polygons, then Phong shading polygons, and finally anti-aliasing. Performance is enhanced by each phase using results from previous phases and trimming the amount of data needed by the next phase. In this way, only a fraction of the pixels in an image may be Phong shaded while the rest may be Gouraud or flat shaded. Similarly anti-aliasing is performed only on pixels around which there is significant color change. The system features fast response to user intervention, encourages user intervention at any moment, and makes useful the idle cycles in a personal computer.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation -- Display algorithms, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism -- Visible line/surface algorithms,

General Terms: algorithms, computer graphics,

Additional Keywords: image synthesis, 3-D rendering, hidden-surface elimination, interaction, anti-aliasing.

1. Introduction

Computer image generation algorithms can be divided roughly into two application areas: interactive and non-interactive. Non-interactive algorithms typically generate high-quality still images or animated sequences, whereas interactive algorithms seek to generate the image in near real-time. In this paper we describe algorithms which unify the two approaches: the best possible image is generated subject to the near real-time constraint, but the image improves with time to a level of quality usually not found in interactive applications.

This paper describes an implementation of a general purpose polygon-based renderer with the above goals. We use the term *refinement* to indicate that the image is constantly being improved, and *adaptive* to indicate that the improvement adapts to the particular nature of the image -- the polygons and pixel locations whose further processing is likely to make the greatest improvement in the picture quality. We have at this time only heuristic measures of picture quality, but hope that the techniques described in this paper will apply even more dramatically with more precise measures of image quality.

2. Related Work

Progressive Transmission of Images

A number of researchers [SLOAN79,KNOWLTON80,HILL83] have addressed the problem of transmitting images over low bandwidth lines. Images are encoded and transmitted so that the user first sees a rough, low resolution representation of the complete image. As time passes, the image is refined until the source image is seen. The user may abort transmission at any time. This provides an efficient means for browsing through an image database.

Our research shares a similar goal: to convey the most information as early as possible, with image quality improving in time. Instead of starting with a complete image at the far end of a communication channel, however, we start with a scene description (e.g., a collection of polygons) and must generate the final image.

Level of Detail Management

The time required to render an image is related to the complexity of the scene description. Two methods have been used to control the size of the rendered data set.

The first method is to generate adaptively the scene description at image generation time based on constraints. Subdivision algorithms [CATMULL74, LANE80, FOURNIER82] are examples of procedure models whose output varies based on view information.

The second method is to store a hierarchy of object descriptions, and choose the most appropriate representation at display time. Clark [CLARK76] recognized the value of this approach, which apparently had been used in flight simulators for many years.

The flight simulator application, however, differs from our application in at least two significant ways. Designing a database for a flight simulator is a one-time task, hence it is reasonable to hand-tune a database. Secondly, many of the objects are procedurally modeled, and can be generated to varying levels of detail by varying the parameters of a single procedure. Medical imaging, one of our applications, must often deal with new data sets that are not procedurally modeled, such as models reconstructed from CT scans. It is possible to generate less detailed versions of these data sets as a post-process [MACDOUGAL84], but in general this is a more difficult problem than generating these versions at original object-definition time (such as with data sets for flight simulators).

Oct-trees [HUNTER78, JACKINS80, MEAGHER80, SAMET84], because of their natural hierarchy, are well-suited for rendering at various levels of refinement; rendering can take place to the desired level of definition. Although we might explore oct-trees for some of our medical applications, we have found polygonal surface representations to be more compact than oct-trees; they also allow very high quality rendering directly.

Multiple Pass Refinement

There are undoubtedly hundreds of implementations of what we will call the two pass approach. In the two pass approach, the user initially manipulates a crude (typically wire-frame) representation of the object and selects a particular view. A more time-consuming, high quality rendering of the scene is then performed. A problem with this approach is that the crude representation may not provide enough information for positioning. Moreover, the higher quality rendering must usually complete in a reasonable amount of time, and hence is frequently not chosen to be of the highest quality possible. The quality levels are determined *a priori* by the graphics system implementor. We prefer to let the user determine when the image is sufficiently detailed.

Forrest [FORREST85] proposed a system with a predefined number of quality levels. A scene is first rendered at level zero, the lowest quality level. If the scene parameters do not change, the scene is then progressively rendered at levels one, two, and so forth. He noted that in some cases the higher levels can be achieved with a small amount of additional computation. The first three of his five quality levels for lines are 1) whatever the hardware can draw, 2) anti-aliased lines without readback, 3) anti- aliased lines with readback. He noted that for smooth-shaded 3-D objects anti-aliasing is needed only in silhouette edges. In addition, he presented a fact that forms a foundation for our work; if the user is working on a personal workstation, there is no reason not to take advantage of idle CPU cycles to improve the quality of the image. We are extending this approach to allow automatically a continuum of quality levels without user definition, to allow efficient image improvement by cutting down on the amount of data remaining for each successively higher level routing (flat, Gouraud, Phong shaders), and suggest refinement methods without any fixed highest level.

Numerous rendering systems first generate a low resolution image, then refine it. For instance, UNC graduate student Andrew Glassner has implemented arbitrary slicing through a 3D density distribution (typically an anatomical structure described by a stack of CT images), by generalizing Crow's sum tables [CROW84]. His algorithm first generates a crude 64x64-pixel image, then refines it to 128x128, then refines that to 256x256 pixels. In general, rendering algorithms that compute pixels independently (e.g., ray tracers) are well suited to this type of operation, while others such as list priority algorithms are not.

The UNC Pixel-Planes project [FUCHS85] proposed a multiple pass approach to anti-aliasing. The image is initially sampled at display resolution. In subsequent passes, the image is also sampled at display resolution, but with samples taken at subpixel offsets from the original samples. The samples at the end of each pass are merged with previous samples to form an anti-aliased image.

An alternative approach to anti-aliasing is described in [BLOOMENTHAL83]. Edges detected using image-processing methods are smoothed during a post-processing step. This post-processing edge inference is useful because they render a variety of object types (polygons, quadrics, patches) and keeping track of pixels to be anti-aliased is difficult in such circumstances.

Objects in the UNC vibrating ("varifocal") mirror system [FUCHS82, MILLS84] are represented as a large set of points. It was noted that the general position of the object can be determined by a small fraction of the points. The system takes advantage of this by rendering the data set in a random order. If the user moves the joystick, a new image is started. If the joystick remains stationary, the image continues to fill in until the entire data set has been rendered.

Adaptive Rendering

Efficient rendering techniques adapt to scene characteristics to minimize the amount of computation performed. Whitted, Lee, Redner, and Uselton [WHITTED80, LEE85] described techniques for adapting the number of rays cast by a ray-tracer to the complexity of the area being sampled.

Cook [COOK84] described a shading system in which a separate shader may be associated with each surface. This permits arbitrarily complex shading computations to be applied where required, without performing extra computation on surfaces where a very simple model is sufficient. Our method also adapts the renderer to the complexity of the surface, with the binding is done at the individual polygon and pixel level. The decision is made on the fly, depending on time available for the rendering process. A single surface of polygons may have a fraction of its polygons flat-shaded, another fraction Gouraud shaded, and another fraction Phong shaded.

The Brown Graphics Group [STRAUSS84] uses a scene format that can be interpreted by a variety of rendering systems. This permits a different renderer to be used on the same data on different workstations [VANDAM86]. The Cornell testbed [HALL83] similarly permits various rendering modules to be interfaced to a common modeler.

3. Our Methods

Our new methods concentrate on the rendering process itself, rather than on object representation techniques. We do this because our applications of medical imaging and molecular modeling impose severe restrictions on the use of certain techniques such as procedural data set generation and multiple, hierarchical representations.

From the user's perspective, the quality of the image from a standard renderer improves with time roughly as shown in one of the curves in Figure 1. Of course, the "quality" of a synthetic image is not easy to quantify, and we do not propose any metric for it here; we simply note that certain operations improve the quality of the image and certain renderers generate higher quality images than others. Any rendering process improves the image quality in time as the rendering progresses to completion; the image then remains constant. Two problems are immediately apparent:

1) the image may not be useful in early stages when perhaps only a few polygons or a few scanlines have been rendered, and

2) the image doesn't improve (obviously) once the rendering is complete.

Our approach seeks to ameliorate both of these weaknesses. For the early phases of the rendering, we restructure the tasks to display some results (vertices, edges) on the screen as soon as possible. The renderer then begins shading the object by scan-converting the object polygons and optionally, the precomputed shadow polygons. The polygon data structure is traversed more frequently than is usual; it is traversed each time there is either more data to display (for instance, polygon edges instead of merely vertices) or a more refined rendering to be performed (for instance, Gouraud instead of flat shading). Our rendering techniques adapt to the specific data set being rendered by performing operations only on "needed polygons" and "needed pixels." This technique prevents the implementation from becoming simply a sequence of increasingly sophisticated renderers; such a naive structure's execution time would be the sum of the times of all the renderers in the sequence.

The next section details our currently implemented techniques. These techniques address only the first of the two problems listed above, that of generating useful images as early as possible. They do not address, for we are still investigating, the second problem -- the renderer never stops, but "keeps improving the image forever."

We have adopted the following guidelines:

- we try to follow the image quality curves of various renderers (Figure 1) by always doing whatever (we estimate) improves the image most at that particular point in time-- painting another point, another edge, another polygon with Phong highlighting, etc.,

- we start a new image whenever new user input is received in order to achieve fast user interaction,

- we perform only necessary work at each phase before going on to the next: Gouraud shading only those polygons that are not "flat", Phong shading only those polygons that have high specular component, anti-aliasing only pixels around which the color changes significantly,

- we use results from previous phases to reduce calculations in later phases, and

- we aim to use all available CPU cycles (a picture is never finished -- "a renderer's work is never done")

4. Implementation

We first preprocess the three dimensional data set by randomly ordering the polygons and converting to a winged-edge data structure format [BAUMGART75]. From the winged-edge data structure, our testbed builds a polygon list, an edge list, a vertex list, and a vertex normal list. The normals require a separate list, since multiple normals may be associated with each vertex, depending on the polygons (flat and/or smooth-shaded) to which the vertex belongs. This format enables fast traversal of the data set for vertices and points, and also provides necessary connectivity information.

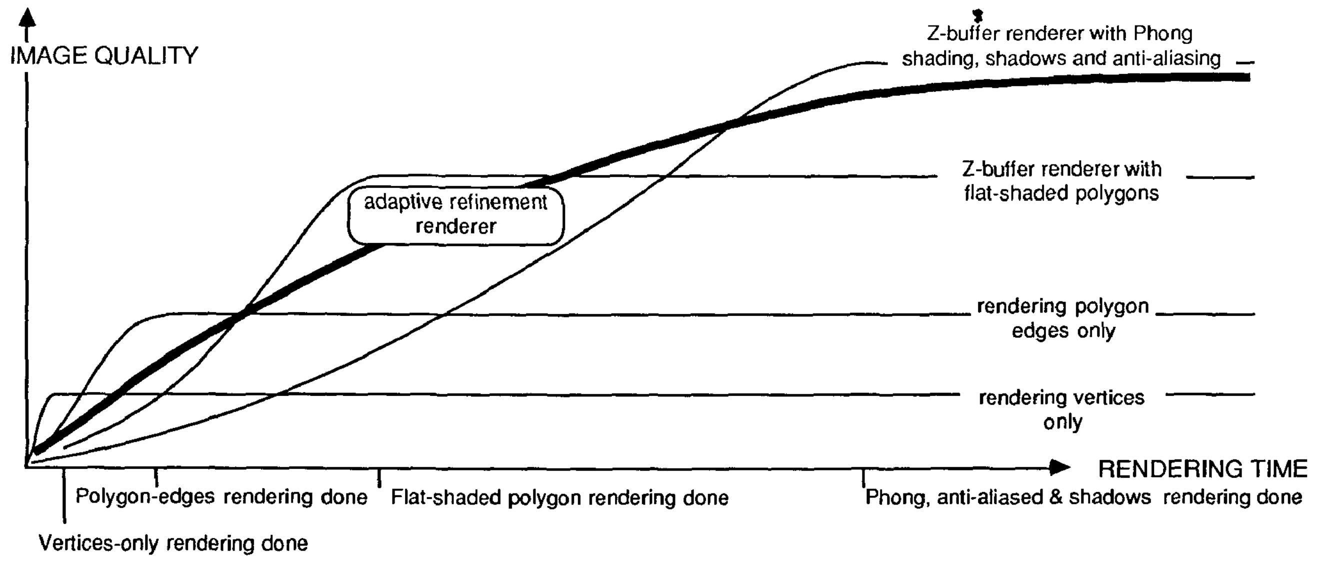


Figure 1: "Image Quality" vs. Rendering Time

In addition to this permanent, viewpoint independent information, our data structure provides storage for values calculated during the rendering process. For example, we save the transformed vertices during vertex display so that transformation need not be repeated during subsequent processing.

Our rendering system provides standard viewing and lighting features. The user can select the view position, viewing angle, hither/yon clipping planes, and light source position, along with the user-specified rendering options detailed in the steps below. The renderer displays shadows if precomputed shadow polygons are provided in the original data set.

Our adaptive rendering system proceeds as follows (Figure 2):

1) **Vertex display** We transform the vertices, build the 6-bit, three-dimensional Cohen-Sutherland clip code [NEWMAN73], and display (with depth cueing) the resulting visible points.

2) **Edge display** We clip the edges of the visible polygons as line segments. The visible edges are displayed with depth cueing using the workstation hardware vector draw.

3) **Flat shading** We complete polygon clipping by joining the clipped edge segments. We then scan-convert the polygons with flat shading using a Z-buffer for hidden surface elimination. A polygon identifier buffer is built to identify the polygon visible at each pixel. This buffer is used by later scan conversion and shadow processing phases. The user may set parameters that control the display of back facing polygons and dithering of color values at each pixel (to enhance the display on workstations with a limited number of colors).

4) **Shadow display** If the data set contains precomputed shadow polygons, we scan-convert these shadow polygons. A pixel is in shadow if the shadow polygon identifier matches the value stored in the polygon identifier buffer. We attenuate the intensity of that pixel by a user specified scaling factor. We also mark that the pixel is in shadow in a pixel attribute buffer (a bit-map which may be stored in either main or non-displayable frame buffer memory). This allows us to adjust the shadow intensity during later processing.

5) **Gouraud shading** We perform Gouraud shading only on those polygons where the range of intensity of a polygon's vertices exceeds a user-specified threshold. We include only the ambient and diffuse components of our lighting model at this stage. Since visibility has already been performed at earlier stages, the process here merely checks the polygon identifier buffer to determine whether or not to display the current polygon at a particular pixel.

6) **Phong shading** We perform Phong shading on those polygons for which the direction of the specular reflectance (highlight) vector at any vertex is within a user-specified tolerance of the direction of maximum highlight. Ideally, this threshold is chosen so that Phong shading is performed only on polygons with noticeable highlights.

7) **Anti-aliasing** We compute a threshold pixel-map that designates which pixels need to be anti-aliased. A pixel is anti-aliased if the maximum range of variance in intensity in the three by three pixel neighborhood around it exceeds a pre-defined threshold. We then build polygon fragments for the designated pixels and perform anti-aliasing with the A-buffer hidden surface algorithm [CARPENTER84].

We note that with some enhancements to the implementation, the above order could be modified under user control.

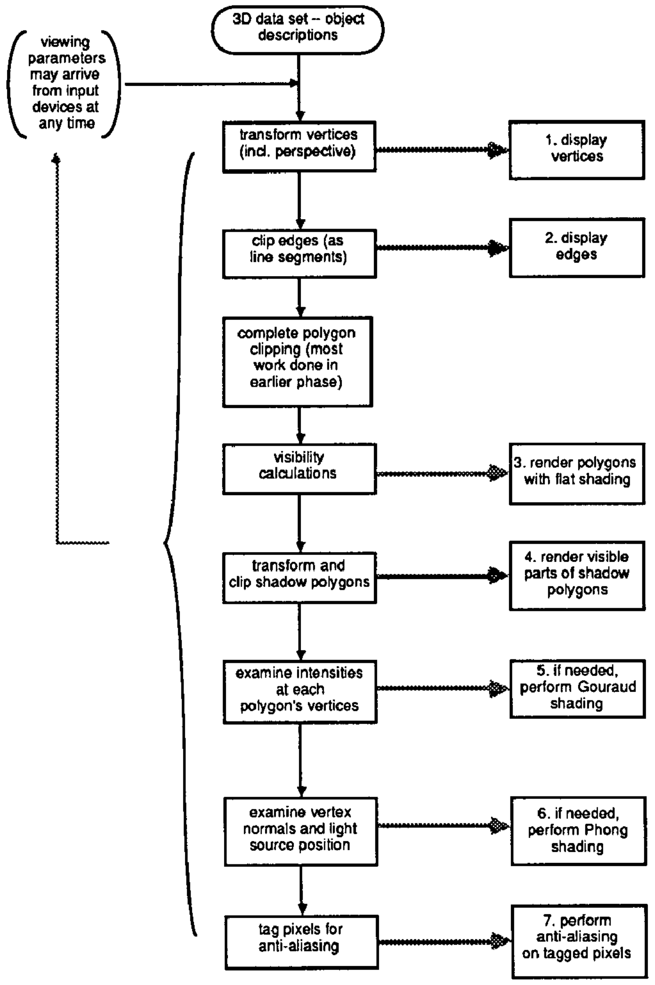

3D data set – object descriptions
viewing parameters may arrive from input devices at any time
transform vertices (incl. perspective)
1. display vertices
clip edges (as line segments)
2. display edges
complete polygon clipping (most work done in earlier phase)
visibility calculations
3. render polygons with flat shading
transform and clip shadow polygons
4. render visible parts of shadow polygons
examine intensities at each polygon's vertices
5. if needed, perform Gouraud shading
examine vertex normals and light source position
6. if needed, perform Phong shading
tag pixels for anti-aliasing
7. perform anti-aliasing on tagged pixels

Figure 2: Rendering pipeline modified for adaptive refinement

5. Experimental Results

Since this conference paper is but the first report on this work, the results thus far are very tentative. However, we already find useful the display of even a small fraction of the vertices whenever these points are updated fast enough for perceptible motion.

We have also found useful the partially completed images early in the rendering process, with polygon edges and a few flat-shaded polygons -- such as the scene illustrated in Figure 3.

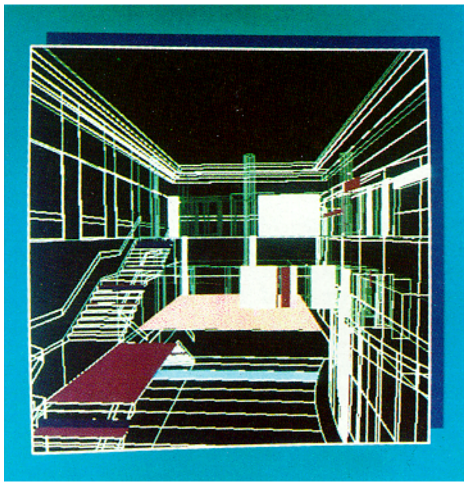


Figure 3: Image in early rendering phase, showing polygon edges and a few flat-shaded polygons (view of lobby of UNC's future Computer Science building).

We have been surprised that the threshold settings for Gouraud and Phong shading have such a major effect on the amount of work being done; the fraction of pixels displayed with Phong shading can easily vary from 5 to 50% depending on the threshold setting. Unfortunately we have not yet had time to characterize these results. We have already found, however, that since the light source direction strongly effects the number of pixels needing Phong shading for specular light component calculations, the user needs to be able to control easily and interactively the location and orientation of the light source in the three-dimensional scene.

The software runs on three different configurations, all running under UNIX: a Masscomp MCS 500 workstation, a DEC VAX-780, and an HP-9000 Series 500. All images except Figures 14 through 19 were photographed directly from the Masscomp screen. The execution times vary from one frame per second (user set) for a fraction of the vertices to about 20 minutes for the Phong shaded image. Figures 14 through 19 show images generated from the implementation running on the HP-9000.

6. Extensions

We hope to add textures and transparency as a later phase similar to the one just described for shadows. We are also eager to try these techniques with some high-speed hardware to see if they could apply in a more structured setting than in our current general purpose workstation environment. In particular, we hope to use some of these techniques on the Pixel-planes system currently under construction [POULTON85, FUCHS85].

Our ultimate goal in this exploration is to find what we call a "golden thread", a single step that if repeated a few times will generate a crude image, one which repeated many times will generate a high quality image, and one which could be repeated indefinitely to yield ever higher quality images. Our current hope is that some form of ray-tracing [WHITTED80,LEE85] with an ever increasing number of rays (and ever deeper "ray" trees) will yield such a "golden thread." Although these notions are purely speculative at the moment, we're encouraged that some of the current calculations may prove useful: the initial ray casting is essentially done by the Z-buffer, and the anti-aliasing bit-map indicates pixels likely to need the most rays.

7. Conclusions

Image generation by adaptive refinement provides an encouraging combination of rendering speed, user convenience, and high quality pictures. The major cost appears to be the large amount of main memory required to store the data set and the intermediate values. With increasing memory capacity for workstations, however, the techniques presented in this paper should become increasingly easy to adopt. Indeed, with more personal, dedicated workstations having spare computing cycles, one may yet develop a renderer whose work is never done.

Acknowledgements

We thank Fred Brooks, Stephen Pizer, and Turner Whitted for many stimulating and useful discussions and for their leadership of many cooperating projects. We thank Julian Rosenman, Edward Chaney, and George Sherouse of the UNC School of Medicine, Department of Radiology for the CT data of the head, and Brian Whitney and Sukumar Ramanathan for generating the polygon description of the Head from the CT data. We thank Dana Smith and Greg Abram for the building data.

We gratefully acknowledge the support of this work in part by the Defense Advanced Research Projects Agency Contract DAAG29-83-K-0148 (monitored by the US Army Research Office, Research Triangle Park, NC), the National Institutes of Health Grant R01-CA39060, and the National Science Foundation Grant ECS-8300970.

References

[ATHERTON78] Atherton, Peter, Kevin Weiler, and Donald Greenberg *Polygon Shadow Generation.* **Computer Graphics**, 12, No. 3 August 1978 pp. 275-281.

[BAUMGART75] Baumgart, Bruce G. *A Polyhedron Representation for Computer Vision.* NCC **1975**, pp. 589-596.

[BLOOMENTHAL83] Bloomenthal, Jules *Edge Inference with Applications to Antialiasing.* **Computer Graphics**, 17, No. 3 July 1983 pp. 157-162.

[CARPENTER84] Carpenter, Loren *The A-buffer, an Antialiased Hidden Surface Method.* **Computer Graphics**, 18, No. 3 July 1984 pp. 103-108.

[CATMULL74] Catmull, Edwin E. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Ph.D. Diss. University of Utah December 1974.

[CLARK76] Clark, James H. *Hierarchical Geometric Models for Visible Surface Algorithms.* **Communications of the ACM**, 19, No. 10 October 1976 pp. 547-554.

[COOK84] Cook, Robert L. *Shade Trees.* **Computer Graphics**, 18, No. 3 July 1984 pp. 223-231.

[CROW84] Crow, Franklin C. *Summed-Area Tables for Texture Mapping.* **Computer Graphics**, 18, No. 3 July 1984 pp. 207-212.

[FORREST85] Forrest, A.R. *Antialiasing in Practice* in **Fundamental Algorithms for Computer Graphics**, Ed. Earnshaw, R.A. in Proc. of NATO ASI Series. Springer-Verlag, 1985 pp. 113-134.

[FOURNIER82] Fournier, Alain, Don Fussell, and Loren C. Carpenter *Computer Rendering of Stochastic Models.* **Communications of the ACM**, 25, No. 6 June 1982 pp. 371-384.

[FUCHS82] Fuchs, H., S.M. Pizer, E.R. Heinz, L.C. Tsai, and S.H. Bloomberg *Adding a True 3-D Display to a Raster Graphic System.* **IEEE Computer Graphics and Applications**, 2, No. 7 September 1982 pp. 73-78.

[FUCHS85] Fuchs, Henry, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton *Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel- Planes.* **Computer Graphics**, 19, No. 3 July 1985 pp. 111-120.

[HALL83] Hall, Roy A., and Donald P. Greenberg *A Testbed for Realistic Image Synthesis.* **IEEE Computer Graphics and Applications**, 3, No. 8 November 1983 pp. 10-20.

[HILL83] Hill, F.S., Jr., Sheldon Walker, Jr., and Fuwen Gao *Interactive Image Query System Using Progressive Transmission.* **Computer Graphics**, 17, No. 3 July 1983 pp. 323-330.

[HUNTER78] Hunter, G.M. *Efficient Computation and Data Structures for Graphics.* Ph.D. Diss. Princeton University 1978.

[JACKINS80] Jackins, C., and Tanimoto, S.L. *Oct-trees and Their Use in Representing Three-Dimensional Objects.* **Computer Graphics and Image Processing**, 14, No. 3 November 1980 pp. 249-270.

[KNOWLTON80] Knowlton, Ken Progressive *Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes.* **Proceedings of the IEEE**, 68, No. 7 July 1980 pp. 885-896.

[LANE80] Lane, Jeffrey M., Loren C. Carpenter, James F. Blinn, and Turner Whitted *Scan Line Methods for Displaying Parametrically Defined Surfaces.* **Communications of the ACM**, 23, No. 1 January 1980 pp. 23-34.

[LEE85] Lee, Mark E., Richard A. Redner, and Samuel P. Uselton *Statistically Optimized Sampling for Distributed Ray Tracing.* **Computer Graphics**, 19, No. 3 July 1985 pp. 61-67.

[MACDOUGAL84] MacDougal, Paul D. *Generation and Management of Object Description Hierarchies for Simplification of Image Generation.* Ph.D. Diss. Ohio State University August 1984.

[MEAGHER80] Meagher, D. *Octree: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer.* Technical Report IPL-TR-80-111. Rensselaer Polytechnic Institute. 1980.

[MILLS84] Mills, Peter H., Henry Fuchs, and Stephen M. Pizer *High-Speed Interaction on a Vibrating Mirror 3D Display.* **Proceedings of SPIE**, 507 August 1984 pp. 93-101.

[NEWMAN73] Newman, William M. and Robert F. Sproull **Principles of Interactive Computer Graphics** 1st Edition, McGraw-Hill 1973 pp. 123-124.

[POULTON85] Poulton, John, Henry Fuchs, John D. Austin, John G. Eyles, Justin Heinecke, Cheng-Hong Hsieh, Jack Goldfeather, Jeff P. Hultquist, Susan Spach *PIXEL-PLANES: Building a VLSI-Based Graphic System* **Proceedings of the 1985 Chapel Hill Conference on VLSI** Computer Science Press pp. 35-60.

[SAMET84] Samet, Hanan *The Quadtree and Related Hierarchical Structures.* **ACM Computing Surveys**, 16, No. 2 June 1984 pp. 187-260.

[SLOAN79] Sloan, Kenneth R., Jr., and Steven L. Tanimoto *Progressive Refinement of Raster Images.* **IEEE Transactions on Computers**, c-28, No. 11 November 1979 pp. 871-874.

[STRAUSS84] Strass, P., M. Shantis, and D. Laidlaw *SCEFO: A Standard Scene Format for Image Creation and Animation.* Brown University Graphics Group Memo, 1984.

[VANDAM86] Van Dam, A. *Personal communication.* 1986.

[WHITTED80] Whitted, Turner *An Improved Illumination Model for Shaded Display.* **Communications of the ACM**, 23, No. 6 June 1980 pp. 343-349

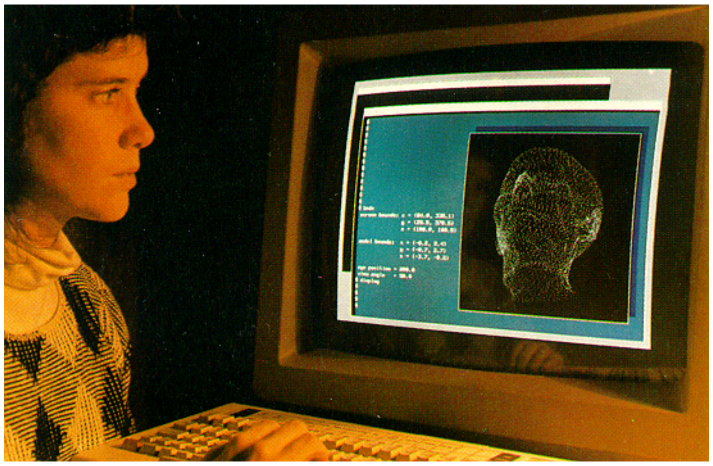
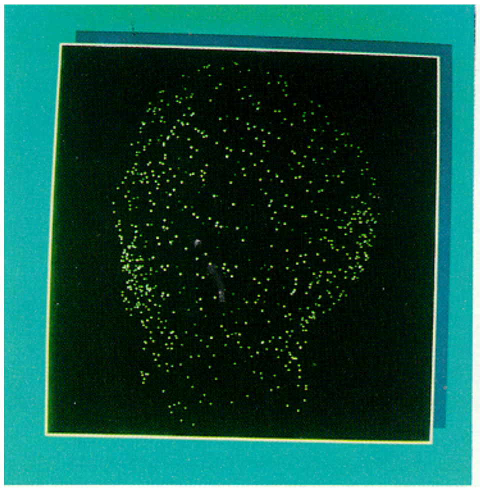


Figure 4: User with image after vertex-display phase.



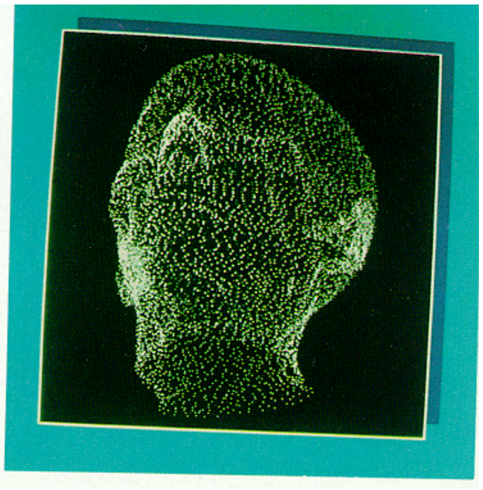


Figure 5: 10% of vertices displayed at ~1Hz. update rate.

Figure 6: All 8,029 vertices displayed.

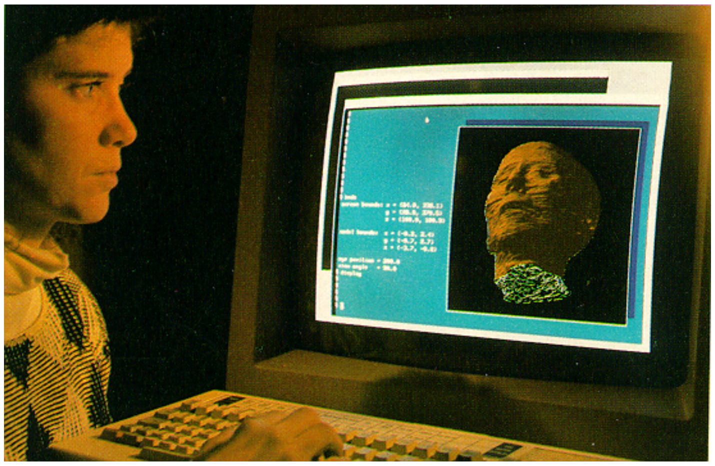


Figure 7: User with image after "flat-shaded polygons" phase.

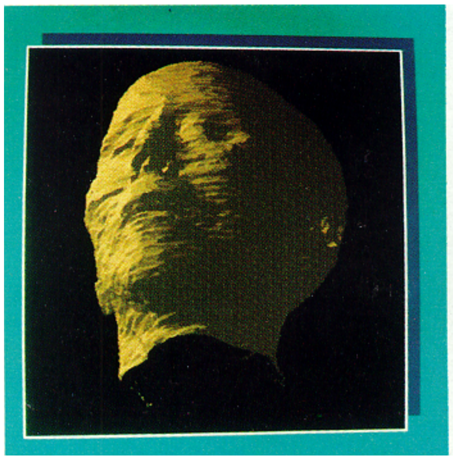


Figure 8: Image after "flat-shaded polygons" phase.

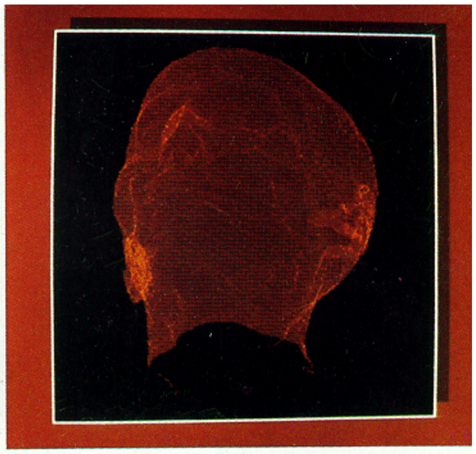


Figure 9: "Cost image" showing computation cost at each pixel for generating Figure 8.

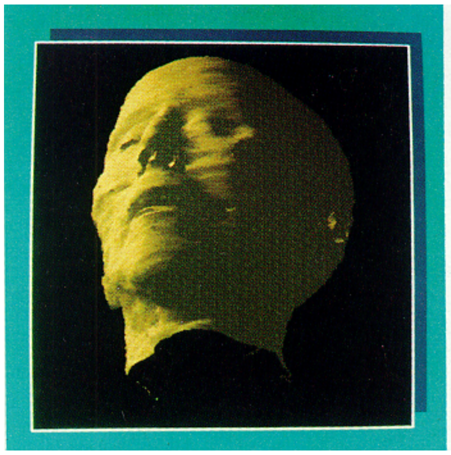


Figure 10: Image after "Gouraud-shaded polygons" phase.

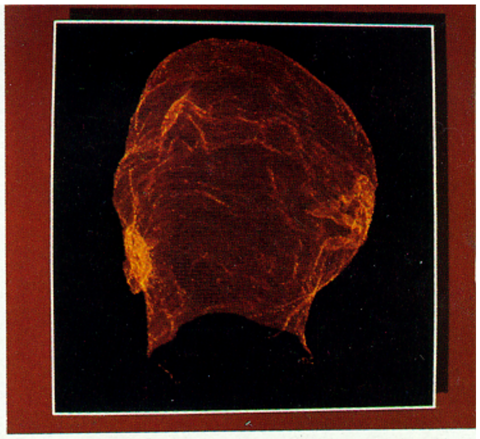


Figure 11: "Cost image" showing computation cost at each pixel for generating the image of Figure 10. Recall that Gouraud shading is performed on a polygon only if the intensity difference among its vertices exceed the user-defined threshold.

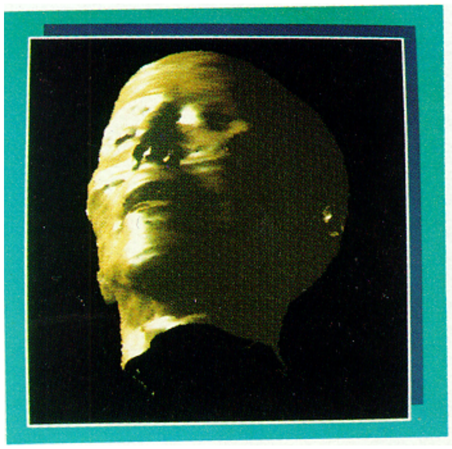


Figure 12: Image after "Phong-shaded polygons" phase.

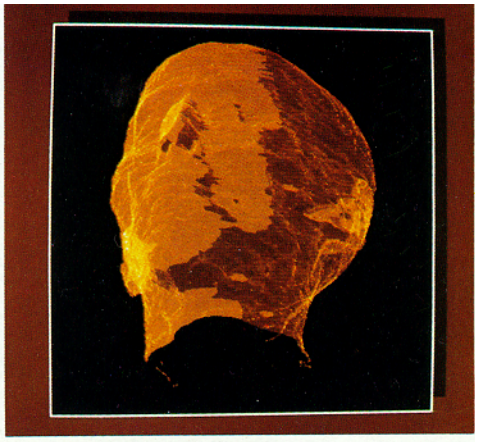


Figure 13: "Cost image" showing the computation expense at each pixel for generating Figure 12. In this case, the user-defined threshold for proceeding with Phong shading passed about 50% of the front-facing polygons.

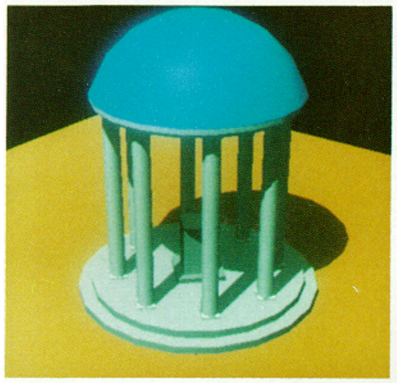


Figure 14: Rendering of the UNC "Old Well" through the Phong-shading phase. 125,459 pixels rendered within the 400 x 400 pixel image.

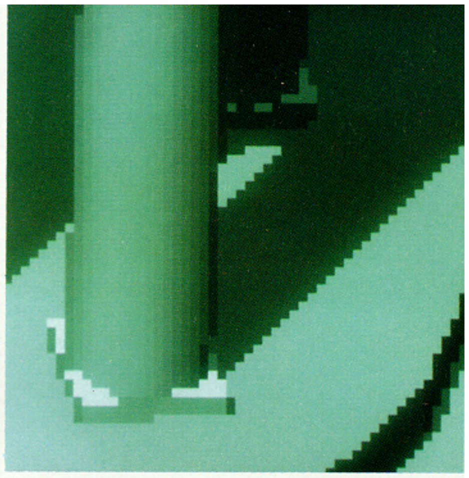


Figure 15: Extreme closeup of one of the column bases showing the deleterious effects of simple, non-anti-aliased rendering.

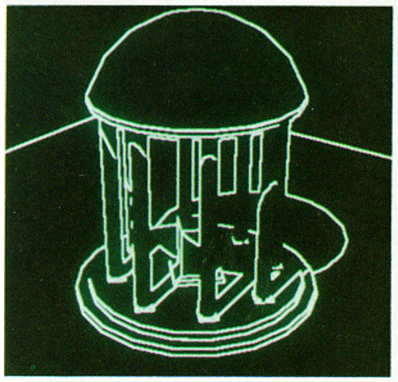


Figure 16: Anti-aliasing map of Figure 14, showing the 13,037 pixels (about 10% of those originally rendered) that will receive anti-aliasing computations.

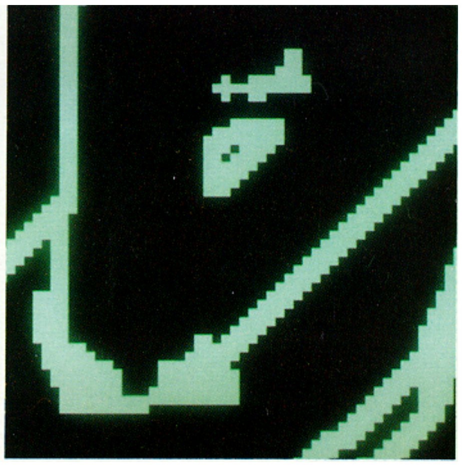


Figure 17: Extreme closeup of the anti-aliasing map, showing same area as Figure 15.

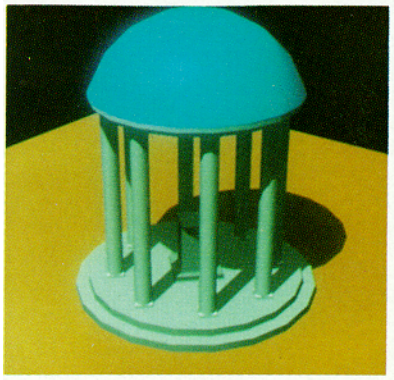


Figure 18: Image of Figure 14 after anti-aliasing computations on the pixels shown in Figure 16. These computations generated 50,722 fragments for the A-buffer -based anti-aliasing.

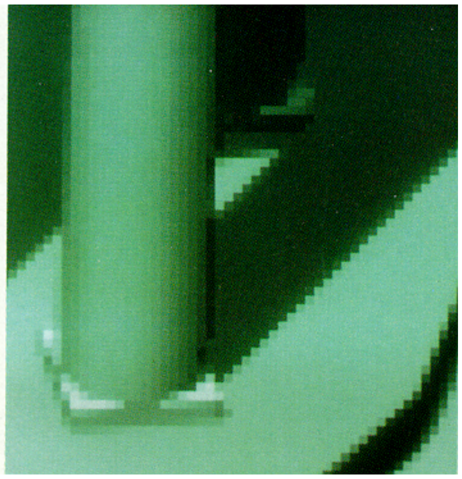


Figure 19: Extreme closeup of Figure 18, showing the same area as Figures 15 and 17.