2004

Reflection Morphing

Andrew Martin

Voicu Popescu
Purdue University, popescu@cs.purdue.edu

Report Number:
04-015

REFLECTION MORPHING

Andrew Martin
Voicu Popescu

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-015
April 2004

Reflection Morphing

Andrew Martin, Voicu Popescu
Computer Science, Purdue University

Abstract

Most scenes of interest to computer graphics applications contain reflective surfaces. Rendering such surfaces accurately and effectively is challenging. Most interactive graphics applications render reflections using environment mapping. Environment mapping approximates reflections drastically. In some situations, a reflected point is drawn hundreds of pixels away from its correct location. Environment mapping does not provide motion parallax, in other words, distant and near reflected objects do not move with respect to each other as the desired view translates. Accurate reflections can be obtained by ray tracing but that comes at the price of sacrificing interactivity.

We describe reflection morphing, a novel algorithm that renders accurate reflections on general reflectors at interactive rates. The difficulty in rendering reflections comes from the fact that a reflected vertex cannot be projected onto the desired view. Reflection morphing builds a set of ray octrees as a preprocess, which are then used at run time to morph the projection of reflected vertices. Once the projection problem is overcome, reflection morphing takes advantage of existing powerful feed-forward graphics hardware. The method provides correct motion parallax and supports moving objects, multiple reflectors, and higher order reflections.

1. Introduction

A major goal of computer graphics is to produce, at interactive rates, images that depict complex three-dimensional scenes with photograph-like fidelity. Research in global illumination and, more recently, in image-based modeling and rendering (IBMR) has produced modeling and interactive-rendering systems that accurately handle *static* scenes in which all surfaces are assumed to be *diffuse*. Such surfaces have the same appearance in any new view requested by the user so the system can efficiently use pre-acquired or pre-computed color samples. Most scenes contain surfaces that cannot be rendered using the diffuse reflection model. An example of such surfaces are reflective, mirror-like, surfaces, which provide important clues to the user exploring the scene. Several techniques have been developed for rendering reflections, each with important limitations. We will review prior work in section 2. The remainder of this section analyzes the problem of rendering reflections and gives a high-level description of the solution adopted by reflection morphing.

1.1. The problem of rendering reflections

The graphics pipeline that proved to be the most appropriate for interactive rendering is the feed-forward rendering pipeline which has two main stages: *transformation*, when the scene primitives are projected onto the desired image plane and *rasterization*, which computes the pixels covered by each primitive. The pipeline is efficient because of two fundamental reasons. First, the transformation stage ensures that from all possible primitives, only those that could affect the output image are considered. Second, the rasterization stage processes all the pixels covered by a given primitive coherently, which amortizes the cost of primitive rasterization setup and allows for incremental computation. In order to use this pipeline for rendering reflective surfaces, the following fundamental challenge has to be overcome.

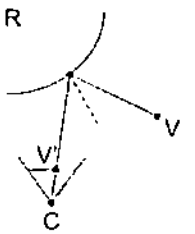

Figure 2 Projection of reflected points

Given a general reflector R (see Figure 2), a desired view C and a point (triangle vertex) V, one cannot easily compute the image plane projection V' of the reflected vertex. For general curved reflectors, or for multiple reflections, there is no closed form solution to the projection equation. Consequently, a triangle whose reflection in R is visible from C cannot be rendered since the image plane coordinates of its vertices cannot be established. We will refer to this problem as the *projection of reflected vertices problem*.

1.2. Reflection morphing idea

Consider a desired view camera that is used to render a scene containing perfect reflectors. The reflectors do not have their own appearance but rather borrow it from the diffuse objects in the scene. Reflectors only affect the direction of a ray, not its color, and should thus be considered as being part of the camera not as part of the scene. Consider the scene in Figure 3. The generalized camera obtained by combining the camera C with the two reflectors can be described as a list of ray segments: (CA_1, CA_2, CA_3, A_1B_1, A_2B_2, A_3B_3, B_1E_0, B_2E_1,

Figure 1 Environment mapping (*top*), Ray tracing (*middle*), Reflection Morphing (*bottom*).

A_1E_4, A_2E_3). The points E are obtained by clipping the rays with a bounding box of the scene. If only the generalized camera could project an arbitrary 3D point efficiently onto its image plane, the problem of projecting reflected vertices would be solved and hardware could take care of rasterization to finish rendering the reflections.

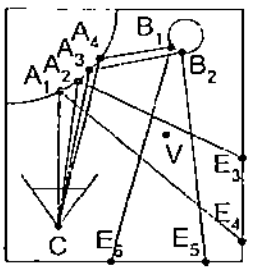

Figure 3 The camera C together with the two reflectors form a generalized camera.

In general, an arbitrary point does not lie exactly on one of the ray segments, so the projection is approximate. Moreover a point can project to multiple locations in the image plane. In Figure 3 the point V projects at A_2 because it is closer to A_2E_3 than to A_1A_4 and at A_4 because it is closer to B_1E_6 than to B_2E_5.

One way of implementing projection for the generalized camera is to traverse the list of rays sequentially, compute the distance from every ray to the given point and return the rays that are closest to the point. Camera C alone has hundreds of thousands of rays, so an acceleration scheme is needed. We use *ray octrees* that subdivide the scene recursively and store at the leaves the subset of rays relevant to that particular region of space. Given a point, the leaf that contains it is located in logarithmic time, and the much shorter ray list stored at the leaf can be searched efficiently to find the closest ray.

Since the desired view changes for every frame, it is impossible to build the ray octree for the desired view. Instead we build ray octrees at the nodes of a regular 3D grid. We call the nodes *sampling locations* and the grid cells *sampling cells*. At run time, the current sampling cell is established using the position of the desired view. Each vertex is projected onto the 8 sampling locations using the ray octrees, and then the resulting 8 projection points are trilinearly interpolated to find the final projection point. Once the projection of the reflected vertex is found, the triangles are rasterized with the help of hardware.

The algorithm computes correct reflections at the 8 sampling locations and then morphs them into the final reflection to take into account the offset within the current sampling cell. The desired view and the diffuse objects can move freely. If the reflectors move, the ray octrees need be recomputed. Reflection morphing renders interactively very high-quality reflections (Figure 1, accompanying video).

2. Previous work

The importance of rendering reflective surfaces has been recognized early on in computer graphics. Phong lighting and shading [Phong 1973] is equivalent to reflecting light sources in shiny surfaces by searching for the appropriate eye-, normal- and light vector combination. If the surface is highly reflective every scene point turns into a light source. It is interesting to note that reflections on arbitrary reflectors could be computed using some hypothetical hardware that supports a very large number of lights. Planar reflectors are rendered by mirroring the real scene across the reflector plane and using stenciling or texturing to confine the reflected world to the surface of the reflector [McReynolds and Blythe 1997].

2.1. Environment mapping

Interactive rendering systems currently approximate reflections on curved reflectors using environment mapping [Blinn and Newel 1976; Greene 1986; Haeberli and Segal 1993; Voorhies and Foran 1994]. The environment map is a spherical or cubical panorama of the scene, rendered from the centroid of the reflector. When the reflector is rendered the eye- and normal- vectors are used to compute the reflected ray which is then looked up in the environment map using *only* its orientation. In other words, the approximation consists of assuming that all reflected rays originate from the same point.

In Figure 4 environment mapping sets pixel P to the color of ray r_e, while the correct color is given by ray r_c. The approximation works well for objects far from the reflector; for nearby objects, the errors introduced are substantial. In Figure 1 the front columns and the cubic particle are close to the surface of the reflective sphere. Ray tracing (*middle*) and reflection morphing (*bottom*) correctly draw the reflections close to the real objects, whereas environment mapping (*top*) fails to convey the proximity of the objects to the surface of the sphere.

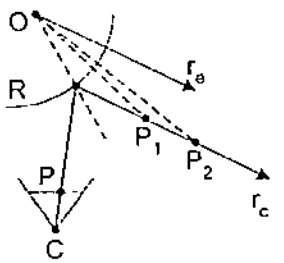

Figure 4 Environment mapping

Another fundamental shortcoming of the method is that it does not provide motion parallax, which is a crucial cue for an explorer of a 3D scene. Referring to Figure 4 again, the points P_1 and P_2 were seen along different rays when the environment map was built and will not have the correct relative movement as the desired view changes. The two points should occlude each other as seen from C, which they will never do, in any environment-mapping rendered reflection. Self-reflections and inter-reflections are also a challenge for the method. The large approximation errors introduced come at no surprise since it essentially recreates novel views of a 3D scene from one image, without using its geometry. We refer the reader to the accompanying video for an illustration of the lack of motion parallax when environment mapping is used.

2.2. Explosion maps

Better results can be obtained by solving the problem of projecting reflected points. Hanrahan and Mitchell describe a search procedure for the projection of reflected points if the reflector surface is given by an implicit equation [Hanrahan and Mitchell 1992]. Ofek and Rappoport introduce the only method that addresses the general case of arbitrary curved reflectors [Ofek and Rappoport 1998; Ofek 1998]. For each reflector, the method computes a *reflection subdivision* consisting of one cell per reflector face. A scene point is projected by first finding the cell that contains it and then using the cell to approximate its projection. Computing and searching the reflection subdivision is accelerated by an approximation, the *explosion map*, which allows for faster indexing. The method has the merit of formulating the problem of general reflections in a form manageable by existing

graphics hardware. The method is expensive as all primitives are considered for all reflectors.

2.3. Reflections in IBMR

The problem of reflections has also been studied by researchers in IBMR. *Light fields* [Levoy and Hanrahan 1996; Gortler et al. 1996] pre-store all rays that could be needed in a database that is queried during rendering. Light fields support view-dependent effects including reflections. However many rays need to be stored to compensate for the lack of geometry. The approach is suitable only for small scenes, typically consisting of one or a few objects (outside-looking-in rendering case). In order to reduce the number of samples needed, IBMR techniques were developed that use some explicit form of geometry and only use samples to capture and render the view dependent effects. *Surface light fields* store all rays originating at each point of a surface, reducing the size of the ray database [Miller 1998; Wood et al. 2000]. In *view dependent texture mapping* photographs of the same surface taken at different angles are blended with weights that measure how similar the current desired view is to each original view [Debevec et al. 1996; Debevec et al. 1998; Pulli et al. 1997]. Both techniques handle surfaces of limited reflectivity well. Highly reflective surfaces require a very dense sampling of the possible view directions which translates into an impractical number of samples.

2.4. Raytracing

Reflections can be computed accurately using ray tracing [Whitted 1980; Glassner 1989], a general technique that produces high quality images. The ray tracing pipeline is less efficient than the feed-forward pipeline because considerable computational effort has to be spent to decide which primitive affects a given output image pixel. Numerous research efforts are targeted at accelerating ray tracing. Wald et al. have demonstrated real-time ray tracing on small scenes on a single general-purpose CPU with vector floating point extensions [Wald et al. 2001a; Wald et al. 2001b]. Off-line ray tracing can be accelerated using commercially available hardware [Hall 2001]. Complex scenes were ray traced at interactive rates on shared memory parallel computers [Parker et al. 1998; Parker et al. 1999] and on clusters [Wald et al. 2001b]. The fixed-function pipeline implemented in commodity graphics accelerators has been replaced with a pipeline that offers programmability both at vertex and fragment level [ATI 2001; Nvidia 2001]. The programs that could originally be executed to process vertices and fragments were too simple to implement ray tracing [Purcell 2002]. Even if the programmability advances sufficiently to allow raytracing, for the foreseeable future GPU's will remain primarily feed-forward rendering engines.

3. Reflection morphing overview

Reflection morphing overcomes the problem of projecting reflected vertices and then takes advantage of the power of feed-forward graphics hardware to render at interactive rates very high-quality reflections. The main steps of the preprocessing stage are:

- Subdivide scene in regular 3D grid that define sampling cells and sampling locations.
- For each sampling location build a *ray octree* that stores at each leaf the list of rays needed for the projection of points located inside the leaf.
- Subdivide scene triangles such that object edges appear correctly curved in the reflected images.

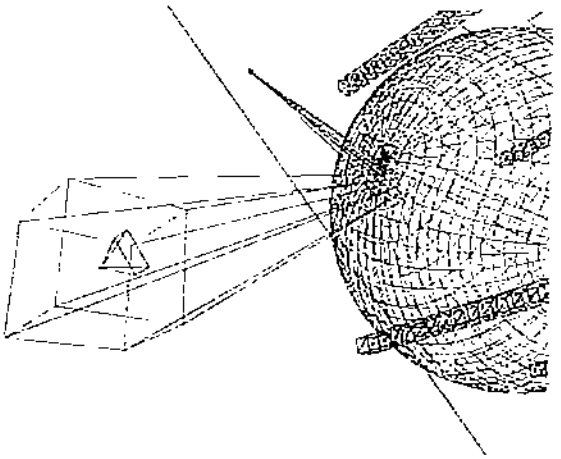

Figure 5 Vertex morphing.

At run time:

- Render diffuse objects
- Render reflector footprints in z and stencil
- Reflection morph diffuse objects over reflector footprints

4. Ray octrees

An octree is built for each sampling location. The algorithm has two main steps.

4.1. Ray maps

Ray maps are layered cube maps that instead of storing color samples, store pointers to rays. Several layers are needed in order to accommodate higher order reflections. Each face and each layer of the ray map is filled in by raytracing the scene without the diffuse objects. This is done in order to conservatively fill in the entire scene space and allow diffuse objects to move from their original position. If some diffuse objects are known to never move, they could be left in which reduces the number of rays. Once the ray maps are filled in, the octree is constructed by recursive subdivision.

4.2. Octree construction

The initial octree is empty and covers the entire scene. Each ray in the ray map is added in turn to the octree. The octree is subdivided if the new ray conflicts with an existing ray. Two rays conflict when they have the same trace. The ray trace is a concatenation of reflector IDs hit by the ray after it left the camera. Only the leaves store rays. When a leaf is subdivided its ray lists are pushed down to the newly created children. A leaf is not subdivided if the maximum octree depth has been reached.

The octree is linearized and saved as a binary file for fast loading.

5. Triangle subdivision

One cannot render the reflection of a large triangle on a curved surface by rendering the triangle formed by its reflected vertices. This would result in straight edges, but the correct reflection would have curved edges. Therefore, the triangles are subdivided. The newly created vertices curve the edge when projected. We subdivide all triangles of diffuse objects to have edges shorter than a certain threshold. We use a regular recursive subdivision scheme. The resulting subdivided triangle meshes are reflection morphed as described below.

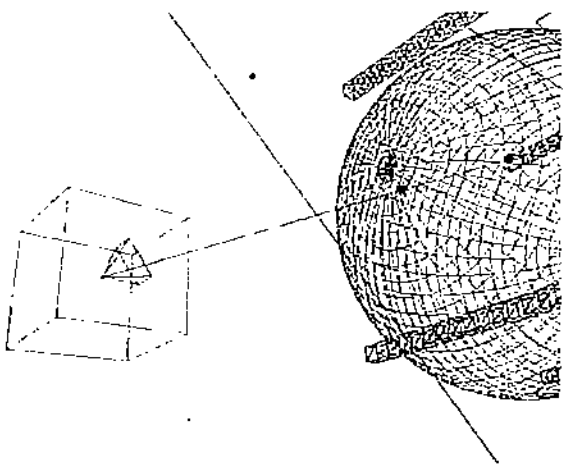

Figure 7 Offsetting of reflected point for depth-buffering.

6. Reflection morphing

The scene rendering algorithm consists of the following steps.

RENDER_SCENE()

- Render the diffuse objects in the scene.
- Render the reflectors in the stencil buffer. A depth test must be enabled during this step. No color needs to be written to the frame buffer for perfect reflectors. This is the place to render the diffuse component if the reflector has one.
- Clear depth buffer. The algorithm will use the depth buffer to ensure that the front object is drawn when the reflections of multiple objects project to the same area of the reflectors.
- Reflection morph each subdivided triangle mesh

REFLECTION_MORPHING()

- The reflection must only be drawn in the region covered by the reflector, so the stencil buffer must be checked.
- Determine the sampling cell occupied by the desired view.
- For every vertex MORPH_VERTEX (please read MORPH_VERTEX section below now)
- For every triangle, if all three morphed vertices are valid, render morphed triangle. Figure 6 shows the reflection mesh obtained by rendering the triangles connecting the morphed vertices.

MORPH_VERTEX

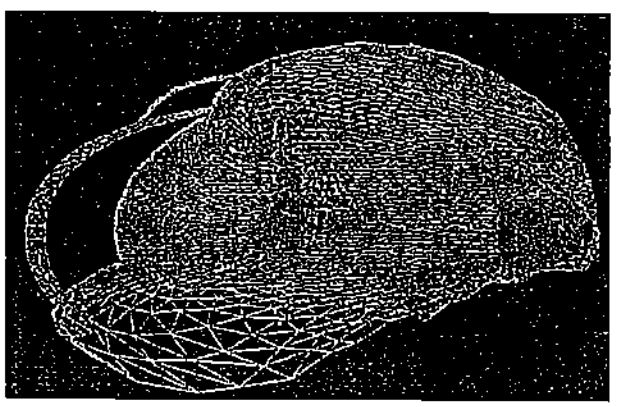

Figure 6 Visualization of the reflection mesh

- Using the ray octrees, look up the points on the reflector where the vertex is seen by each of the eight sampling locations. In Figure 5 the wireframe box shows the current sampling cell.
- Compute the reflected point by trilinearly interpolating these eight points, using the relative position of the desired view within the current sampling cell.
- Push this point along the ray from the desired view to itself (Figure 7). The distance that the point must be pushed is equal to the distance between itself and the actual scene vertex. This will ensure correct depth-buffering since points that are closer to the reflector should occlude the reflection points that are farther away from the reflector. The reader will note that this offsetting does not change its projection onto the desired view.

7. Hybrid reflectors

Not all curved reflectors require reflection morphing. If a reflector is composed of several parts, good results can be obtained from rendering the larger pieces with reflection morphing and the smaller pieces with environment mapping (Figure 8).

8. Results

If all vertices in a scene are dynamic, then each vertex must be morphed for each frame. However, in many cases, much of the scene is static. These vertices must only be morphed once for each sampling cell. This greatly improves performance. We measured a morphing performance of 20,000 dynamic vertices per second and 200,000 static vertices per second. Performance was measured on a Pentium 4 2GHZ, 2GB PC.

Reflections can be rendered with high quality with a small amount of preprocessed data. With a ray map resolution of 256 x 256, an octree depth of 5, and one reflective bounce allowed, the preprocessed data required 4.1 - 5.4 MB of space for a sampling location. A location's average number of rays at a leaf was in the 10.4 – 13.5 range, and its maximum number of rays was in the 227 - 353 range. Preprocessing takes about 3 minutes per sampling location.

9. Discussion and future work

We have presented a technique for rendering curved reflectors that is fast and produces high-quality images. Reflection morphing allows moving diffuse objects and exhibits motion parallax during translation of the desired view. In the future, reflection morphing could be extended to render higher order reflections. Much of the necessary framework for this is already in place. It might also be extended to allow moving reflectors. The technique could also be modified to produce other effects. If the ray octrees contained refracted rays instead of reflected rays, it would be possible to render curved refractive surfaces quickly and correctly.

References

ALIAGA, D. and LASTRA, A. 1999. "Automatic Image Placement to Provide a Guaranteed Frame Rate", *Proceedings of ACM SIGGRAPH*, 307-316.

ATI. 2001. RADEON 8500 product web site. http://www.ati.com/products/pc/radeon8500128

BLINN, J.F. AND NEWELL, M.E. 1976. Texture and Reflection in Computer Generated Images. *C.ACM. 19(10)*, 542-547.

CHANG, C., BISHOP, G. AND LASTRA, A. 1999. LDI Tree: A Hierarchical Representation for Image-based Rendering. *Proc. of SIGGRAPH '99*, 291-298

CHEN, E., WILLIAMS, L. 1993. View Interpolation for Image Synthesis. *Proc. of SIGGRAPH '93*, 279-288.

DEBEVEC, P., TAYLOR, C., MALIK, J. 1996. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Computer Graphics, SIGGRAPH 96* Proceedings, 11–20.

DEBEVEC, P., YU, Y., AND BORSHUKOV, G. 1998. Efficient view-dependent image-based rendering with projective texture-mapping. In *Proceedings of the 9th Euro-graphics Workshop on Rendering*, 105–116.

HALL, D. 2001. The AR350: Today's ray trace rendering processor. 2001 *SIGGRAPH /* Eurographics Workshop On Graphics Hardware Hot 3D Session 1. http://graphicshardware.org/previous/www2001/ presentations/Hot3D Daniel Hall.pdf.

HANRAHAN P. AND MITCHELL, D. 1992. Illumination from curved reflectors. *Proceedings SIGGRAPH '92*, ACM Press, pp. 283 291.

GLASSNER, A. 1989. An introduction to ray tracing. *Academic Press*.

GORTLER, S., GRZESZCZUK, R., SZELISKI, R.,AND COHEN, M. 1996. The Lumigraph. *Proc. of SIGGRAPH '96*, 43-54.

GREENE, N. 1986. Environment mapping and other applications of world projections. *IEEE CG&A*, 6(11).

HAEBERLI, P. AND SEGAL, M. 1993. Texture mapping as a fundamental drawing primitive. *Proceedings, Fourth Eurographics Workshop on Rendering*, Cohen, Puech, Sillion (eds), 259-266.

LEVOY, M. AND HANRAHAN, P. 1996. Light Field Rendering. *Proc. of SIGGRAPH '96*, 31-42.

MAX, N. AND OHSAKI, K. 1995. "Rendering Trees from Precomputed Z-Buffer Views", Rendering Techniques '95: *Proceedings of the 6th Eurographics Workshop on Rendering*, 45-54.

MCMILLAN, L. 1997 An Image-Based Approach to Three-Dimensional Computer Graphics. *PhD thesis, University of North Carolina at Chapel Hill.*

MCREYNOLDS, T. AND BLYTHE, D. 1997. Programming with OpenGL: Advanced Rendering. *SIGGRAPH '97* course. SGI Copyright ©1997 www.sgi.com/software/opengl/advanced97/notes/node89.html

MILLER, G. et al. 1998. "Lazy Decompression of Surface Light Fields for Precomputed Global Illumination", *Eurographics Workshop on Rendering 1998*, Vienna, Austria, June 29th - July 1st.

NVIDIA GeForce3 Ti Family: Product overview. 2001. 10.01v1. http://www.nvidia.com/docs/lo/1050/SUPP/gf3ti overview.pdf.

OFEK, E. AND RAPPOPORT, A. 1998. Interactive reflections on curved objects. *Proceedings SIGGRAPH '98*, ACM Press, 333-342.

OFEK, E. 1998. Modeling and rendering 3-D Objects. Ph.D. thesis, Institute of Computer Science, The Hebrew University.

PARKER, S. et al. 1998. Interactive ray tracing for isosurface rendering. In IEEE Visualization '98, 233–238.

PARKER, S. et al. 1999. Interactive ray tracing. In 1999 ACM Symposium on Interactive 3D Graphics, 119–126.

PHONG, B.T. 1973. Illumination for Computer-Generated Images Ph.D. Dissertation, Department of Computer Science, University of Utah, Salt Lake City.

POPESCU, V. 2000. The WarpEngine: An Architecture for the Post-Polygonal Age. *Proceedings of SIGGRAPH 2000*, ACM Press, pp 433-442.

PULLI, K., COHEN, M., DUCHAMP,T.,HOPPE, H., SHAPIRO, L., AND STUETZLE, W. 1997 View-based rendering: Visualizing real objects from scanned range and color data. In *Eurographics Rendering Workshop 1997*, 23 -34

PURCELL, T.J. et al. 2002. Ray Tracing on Programmable Graphics Hardware, *Proceedings of SIGGRAPH 2002*.

SHADE, J., GORTLER, S., HE, L., AND SZELISKI, R. 1998. Layered Depth Images. *Proc. of SIGGRAPH '98*, 231-242.

SEITZ, S.M. AND DYER, C.R. 1996. View Morphing. Proceedings, *SIGGRAPH '96*, ACM Press, 21-30.

SIMMONS, N. and SEQUIN, C. H. 2000 "Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering", *Eurographics Workshop on Rendering 2000.*

VOORHIES, D. AND FORAN, J. 1994. Reflection vector shading hardware. *Proceedings, SIGGRAPH '94*, ACM Press, pp. 163-166.

WALD, I., SLUSSALEK, P., AND BENTHIN, C. 2001. Interactive distributed ray tracing of highly complex models. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, 277–288.

WALD, I. et al. 2001. Interactive rendering with coherent ray tracing *Computer Graphics Forum 20*, 3, 153-164.

WESTOVER, L. 1990 "Footprint Evaluation for Volume Rendering", *Proc. SIGGRAPH '90*, 367-376.

WHITTED, T. 1980. An improved illumination model for shaded display. *Comm. Of the ACM*, 23(6):343-349.

WOOD, D.N. et al. 2000. Surface light fields for 3D photography. *Proceedings, SIGGRAPH '00*, ACM Press, pp. 287-296.

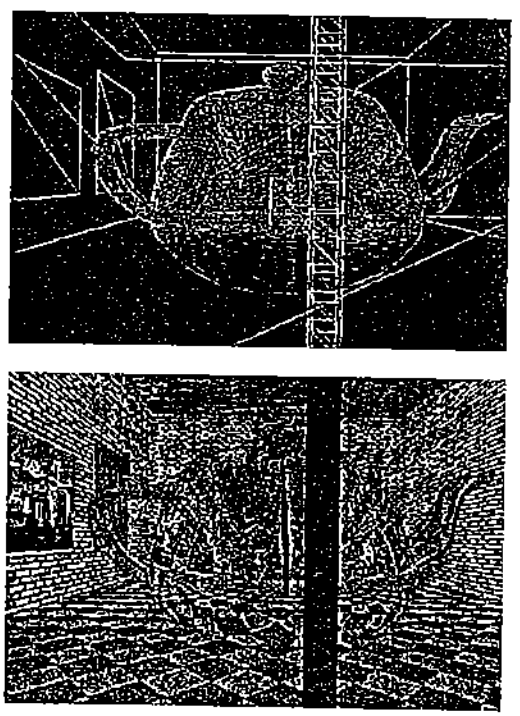

Figure 8 The body of the teapot was rendered with reflection mapping while the spout, handle and lid were rendered with environment mapping.