
10 Interactive Non-Photorealistic Rendering

Bruce Gooch and Amy Ashurst Gooch
Department of Computer Science
University of Utah
<http://www.cs.utah.edu/>

10.1 Introduction

Most of the work in NPR has been static 2D images or image sequences generated by a batch process. In this part of the course notes we explore interactive NPR through the example of interactive technical illustration [4].

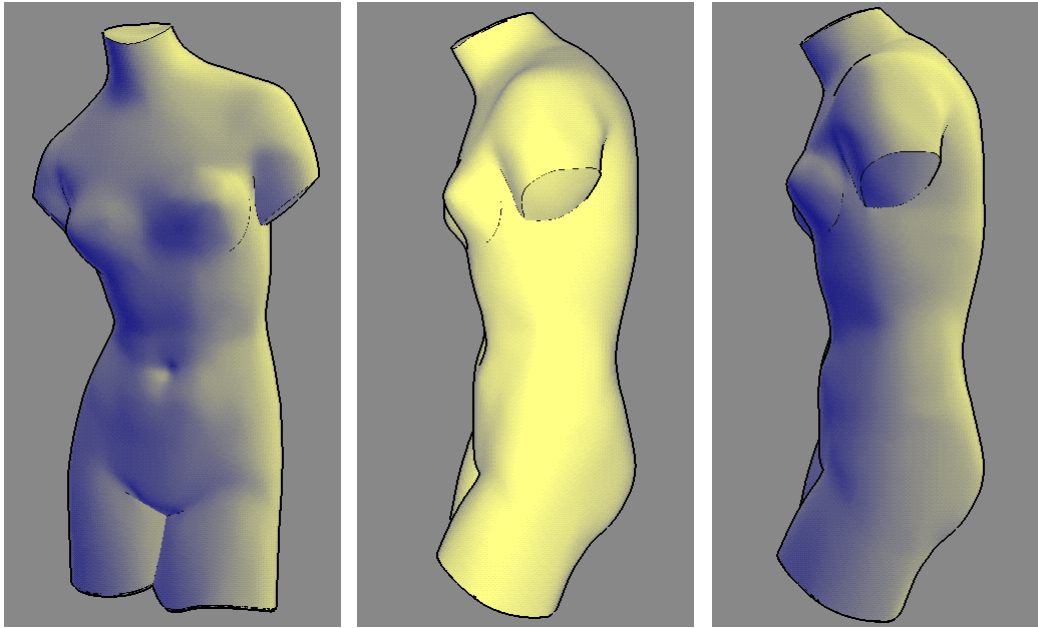
Work that has been done on computer generated technical illustrations has focused on static images and has not included all of the techniques used to hand draw technical illustrations. We present a paradigm for the display of technical illustrations in a dynamic environment. This display environment includes all of the benefits of computer generated technical illustrations such as a clearer picture of shape, structure, and material composition than traditional computer graphics methods. It also takes advantage of the three-dimensional interactive strength of modern display systems. This is accomplished by using new algorithms for real time drawing of silhouette curves, algorithms which solve a number of the problems inherent in previous methods. We incorporate current non-photorealistic lighting methods, and augment them with new shadowing algorithms based on accepted techniques used by artists and studies carried out in human perception.

An interactive NPR system needs the capability to interactively display a custom shading model and edge lines. In addition, this interaction must be possible for complex geometric models. In this part of the course notes we describe a variety of techniques for achieving these goals, and describe the tradeoffs involved in choosing a particular technique.

10.2 Making it Interactive

There are several new issues to address when creating 3D illustrations. Three-dimensional technical illustrations involve an interactive display of the model while preserving the characteristics of technical illustrations. By allowing the user to move the objects in space, more shape information may be available than can be conveyed by 2D images. Interaction provides the user with motion cues to help deal with visual complexity, cues that are missing in 2D images. Also, removing the distracting wireframe lines and displaying just silhouettes, boundaries, and discontinuities will provide shape information without cluttering the screen, as discussed previously in Section 8.

The question remains, “How do the 2D illustration rules change for a 3D interactive technical illustration?” Adapting the shading and line conventions presented previously in the course notes is fairly straightforward as long as the line width conventions have frame-to-frame coherence. The more interesting issues depend upon changing the viewer’s position versus moving the object. Since there are no protocols in traditional illustration, it may be best to model these 3D illustration conventions based on how you would move real object. This has an effect on how the light changes with respect to the object. The light position can be relative to the object or to the viewer. When looking at a small object in your hand, you turn the object and do not move your head, so the light stays in the same position relative to your eye. However when you move an object in an modeling program or when you look at a large part, the view



(a) Model with cool to warm shading with lights positioned up and to the right.

(b) After the camera position is moved to view the side of the model.

(c) After moving the object instead of the camera, allowing the surface to vary completely from cool to warm. (See Color Plate)

Figure 1: Frames from the NPR JOT Program [7], which uses Markosian et al.'s silhouette finding technique [8] and incorporates the OpenGL approximation to the shading model presented by Gooch et al [2].

point is actually moving, not the object. As shown in comparing Figure 1(b) and Figure 1(c), the shading model is used to its full advantage if the surface varies completely from cool to warm.

When illustrators light multiple objects, they may use a different shading across different objects, inferring that each object has its own light, which does not affect the other objects in the environment, similar to the “virtual lights” by Walter et al. [11]. For example, two objects in a scene may be lit differently to draw attention to different attributes of each object. If this was accomplished by adding two lights to the environment, the multiple highlights could be confusing.

Most material properties are semi-constant as the view direction or lighting changes. However the metal shading presented in Section 8 is the replication of the anisotropic reflection [5] due to the surface of the object and the reflection of the environment. When a real metal part is rotated in your hand, the banding does not stick to the object, but remains constant since the environment is not changing. However, in a non-photorealistic interactive environment it may be too jarring to have the metal shading changing abruptly. Using a metal texture might be more appropriate and a metal texture in an interactive environment would still convey the material property.

Another notion is to allow the relative size of the object to control the motion of the viewer, the object, and the light source in an interactive 3D illustration. In the end, allowing the user to choose whether the object moves or the eye point changes, as well as having control over the lights, may help the viewer gain the most shape information.

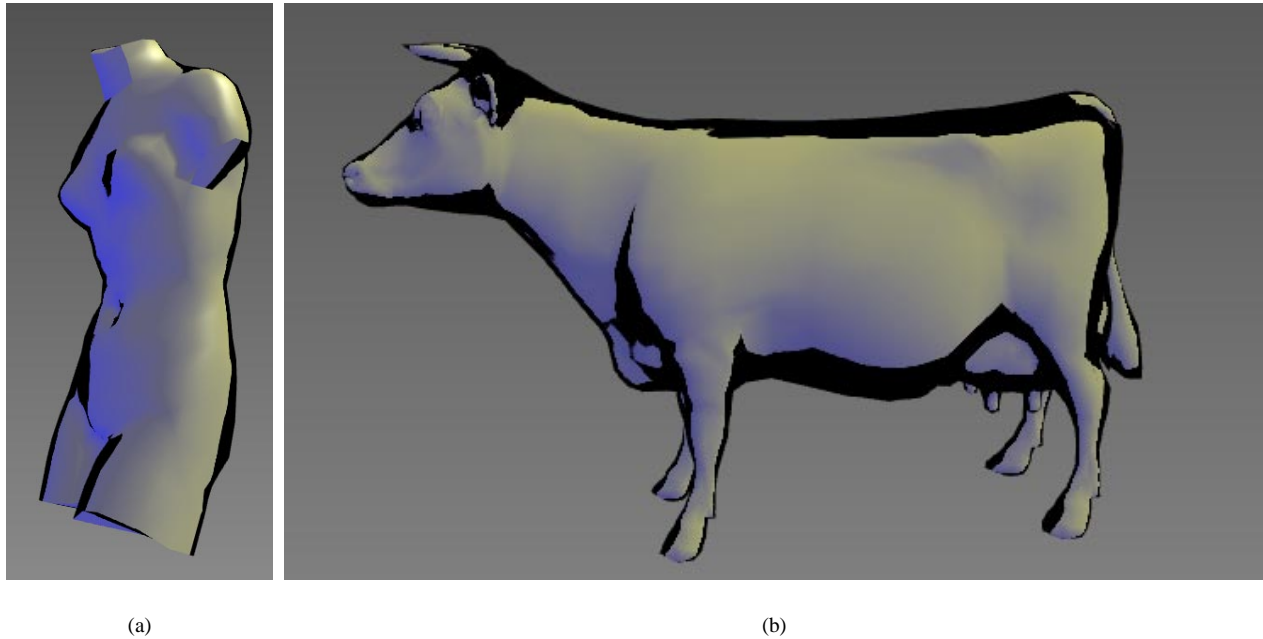


Figure 2: Adding the silhouettes to the environment map instead of calculating silhouettes from the geometry produces interesting artistic effects. (See Color Plate)

10.3 Displaying Important Edges and Silhouettes

Markosian et al. [8] have published real-time software algorithms for drawing edges. Their SIGGRAPH paper is included in these notes, and these methods were used to render Figure 1. Raskar and Cohen [10] have explored the uses of hardware to generate edge lines, similar to the methods we will discuss. We defer to their paper, included at the end of this section, for the details on their algorithms. Interactive technical illustration has also been explored for (Non-Uniform Rational B-Spline) NURBS surfaces [3], which we will not discuss here. To draw silhouettes, we have implemented several methods of extracting and displaying edge lines from polyhedral models, which we discuss in Section 10.3.1 and 10.3.2.

Methods for generating edge lines for polyhedral models can be roughly broken down into two categories. The first assumes no prior knowledge or preprocessing of the model and heavily leverages commodity graphics hardware. The second set of methods use preprocessing of the model and are purely software algorithms. Both hardware and software algorithms clearly have a place. The set of hardware methods are useful because of ease of implementation. The software methods are useful due to their flexibility and lower computational complexity. All of the software methods assume that the models are either manifold or manifold with boundary.

We can also extract boundary curves, edges adjacent to only a single face, and creases, that is, the edge between two front facing polygons whose dihedral angle is above some threshold. The user is provided the option to draw these edges, dependent upon the model and intent of the image. The computation and drawing of creases is discussed in Section 10.4.

10.3.1 Hardware Methods

Using multi-pass rendering [1] there are several ways to extract silhouettes. The algorithm presented in the SIGGRAPH 1998 OpenGL Course does not capture internal silhouette edges and requires four passes of rendering. Below we provide algorithms which require two or three rendering passes and capture internal silhouettes.

In a polyhedral model, a silhouette is an edge that is connected to both a front facing and a back facing polygon. The following is pseudo code for the basic algorithm:

```
draw shaded front faces
draw front faces in line mode:
    setting only stencil
draw back faces in line mode:
    setting color if stencil was set
    decrementing stencil if drawn
```

To draw lines over polygons, the PolygonOffset extension (or PolygonOffset function in GL 1.1) [9] is needed. This function effectively modifies the depth values of the first pass based on the slope of the triangles and a bias factor. This technique can create something similar to a silhouette, effectively a halo. The depth values are pushed forward instead of back to allow lines to be rasterized over faces. Then wide lines are drawn. Where there are large discontinuities in depth (silhouettes and boundaries), only part of the line is drawn. This method requires only two passes instead of the three listed above, but can be fairly sensitive to the parameters of the polygon offset function. Using OpenGL hardware makes the implementation simple, however, it limits the thickness of the edge lines.

Another hardware technique is to add the edge lines to a shading environment map as a preprocess. However, as shown in Figure 2(a), the lines lack crispness, and if the model varies greatly in curvature, there may be large black regions. In order to include silhouettes on the feet of the cow in Figure 2(b), we have to set the threshold low enough to draw lines in these high curvature regions. This causes regions which have relatively low curvature to be filled in with black. Although this effect produces some interesting, artistic results, it may be inappropriate for technical illustration.

10.3.2 Software Methods

A straightforward way to draw silhouettes is to explicitly test every edge in the model. We compute an edge structure based on the face normals of the model, which are also used for back face culling as in Zhang et al. [12]. An edge is a silhouette edge if and only if:

$$(\vec{n}_1 \cdot (\vec{v} - \vec{e})) (\vec{n}_2 \cdot (\vec{v} - \vec{e})) \leq 0,$$

where \vec{v} is a vertex on the edge, \vec{e} is the eye point, and \vec{n}_i are the outward facing surface normal vectors of the two faces sharing the edge. This situation only occurs when one face is front facing and the other is back facing. While this computation is simple, it can potentially become a bottleneck with large models. Since we have to shade (or prime the z buffer for hidden surface elimination) this computation can be done in parallel while the model is being rendered.

We use a more complex preprocess and search algorithm when classifying edges becomes a bottleneck. This algorithm is similar in spirit to Zhang et al. [12], but requires looking at arcs on the Gauss map instead of points. The Gauss map of an edge on a polyhedral model is a great arc on the sphere of orientations (Figure 3). Under orthographic projection, a plane through the origin in this sphere defines the view.

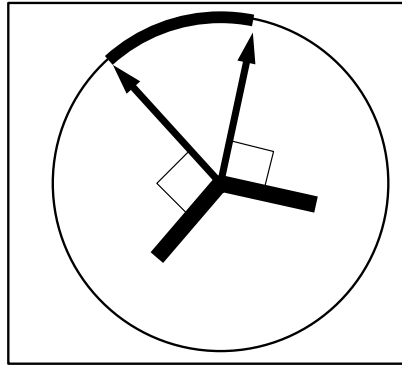


Figure 3: The arc in a Gauss map seen in 2D. The two bold line segments are faces that share a vertex. The orientations of their normals can be represented as points on the circle. The arc between those two points represents all orientations swept out between the two normals. In 3D the same reasoning applies, and the arc is an arbitrary segment on a great circle.

All of the faces on one side of the plane are front facing, and on the other side they are back facing. If the “arc” corresponding to an edge is intersected by this plane, it is a silhouette edge. To search for such edge/plane intersections, we store the arcs in a hierarchy on the sphere to quickly cull edges that can not be silhouettes. We have implemented a decomposition of the sphere starting with a platonic solid (octahedron or icosahedron) and all successive levels are four to one splits of spherical triangles. An arc is stored at the lowest possible level of the hierarchy. This makes silhouette extraction logarithmic in the number of edges for smooth models where the arcs tend to be short. One problem with this hierarchy is that the edges of the spherical triangles on the sphere interfere with the arcs and limit how far they can be pushed down the hierarchy. The probability of being stored in a leaf node that can contain an arc of a given length decreases as the size of the triangles shrink because the boundaries of these spherical triangles become denser as you recurse. An ad hoc solution to this problem is to use multiple hierarchies, whose spherical triangles are different, and store an arc in the hierarchy with the spherical triangle with the smallest area that contains it. A more attractive alternative would be to use “bins” on the sphere that overlap and/or making data dependent hierarchies.

Under perspective viewing, the region you have to check grows, based on planes containing the object and intersecting the eye. Building a spatial hierarchy over the model as in [12] would minimize this effect. One advantage of any software approach is that it makes it easier to implement different styles of line drawing.

10.4 Line Styles

As discussed in Section 8, line width can appear to change by either shading the lines based on the surface orientation, or by using OpenGL 1D texture mapping hardware to shade lines. A 1D texture map can convey the distance between a surface(edge) and a light in the scene.

Fat boundary lines can be drawn with either the software or hardware methods. These lines are drawn after the rest of the model has been drawn (shading, creases, silhouettes). While the earlier phases are drawn, they set a stencil bit, indicating that the given pixel has been draw for this frame. Finally, the boundary silhouettes are drawn over again with wider lines. In hardware this requires a full traversal of the front or back faces, while using software extraction algorithms only require a traversal of the silhouette edges which have been previously computed. All of these algorithms are more efficient than the methods mentioned in the OpenGL course [1] because it required four rendering passes while these algorithms

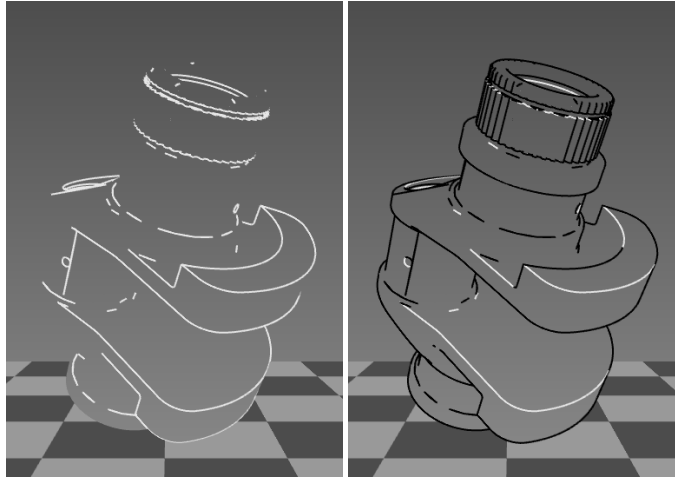


Figure 4: All creases are drawn in white (Left), and then all of the silhouette lines are drawn in black (Right), overlapping the creases.

Model	Faces	Edges	Naive	Gauss	Num Sil
S Crank	24999	35842	.027	.0198	3873
L Crank	169999	254941	.165	.096	12469
Sphere	78804	117611	.082	.016	273

Table 1: Model information and timings, in seconds on 195Mhz R10k for *naive* and *hierarchical silhouette extraction* methods under an orthographic view.

require only one extra pass, and that pass may only be of the silhouette edges.

Creases are extracted independent of the view and are drawn as white lines. After adding shading and silhouettes, only the creases that are connected to two front facing faces, and are not already silhouettes, are visible. To emulate the look of illustrations the creases need to be drawn with the same thickness as the silhouettes, as shown in Figure 4.

One problem when rendering rasterized wide lines is the “gaps” where the lines do not overlap. A solution to this is to render the end of the lines with large points, effectively filling in the gaps. There is much less of a performance loss with the software extraction methods, since they only need to redraw the actual silhouettes, not the entire model.

10.5 Discussion

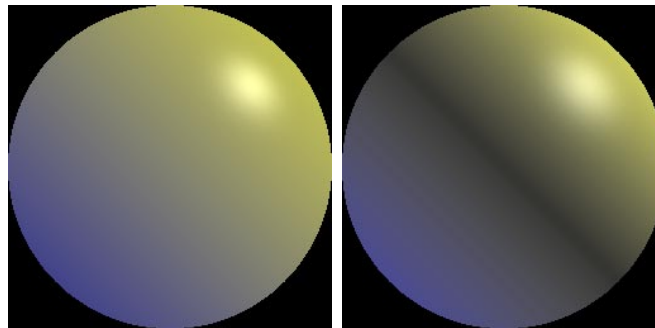
Silhouette finding using specialized graphics APIs like OpenGL is simple to implement and not as dependent on “clean” models. However it is less flexible and does not allow the user to change line weight. The software methods we discussed are more complex and depend on “clean” models which must have shared vertices, otherwise internal boundaries can not be checked for silhouettes. However the software methods provide more flexibility and, potentially, better performance.

Table 1 presents the information of two extreme cases. These cases are based on orthographic views. Under perspective projection some form of bounding volume hierarchy would have to be employed [12] to increase the efficiency. Both the simplified and the finely tessellated versions of the crank shaft model have many sharp features, while the sphere has very small dihedral angles.

The current implementation of the hierarchy method uses an icosahedron with four levels of subdivision which generates 1280 faces. On the sphere this method is extremely efficient. When using overlapping

Level	No Overlap	Overlap
0	12294	3138
1	4844	2221
2	5978	3569
3	4666	5943
4	9704	22615

Table 2: Hierarchy method showing the number of edges stored at each level on a Gaussian sphere for 25k-polygon crank shaft model for non-overlapping and overlapping bins.



(a) Environment map used to generate Figure 6(a).

(b) Environment map used to generate Figure 6(b).

Figure 5: Shaded sphere images used for environment maps. (See Color Plate)

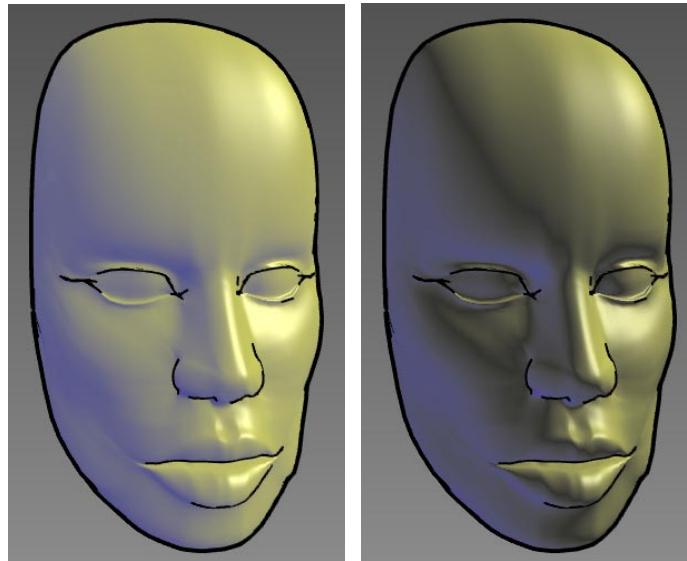
bins, all of the edges are stored in the leaf nodes. When using non-overlapping bins only 84% of the edges are in the leaf nodes and 2132 are on level zero. Table 2 shows the number of edges stored at every level of the hierarchy for non-overlapping and overlapping hierarchies. The overlapping method did a much better job, even on the simplified crank model.

Parallelizing the silhouette extraction with the rest of the rendering can cause the extraction time to be negligible. A separate thread can extract silhouettes while the polygons are being rendered to shade the model or initialize the Z buffer. This parallelization takes only three-thousandths of a second for the sphere and five one-hundredths on the large crank shaft model. If you are using software visibility algorithms this technique would probably prove to be more effective.

10.6 Shading

There are several ways to apply NPR shading models using hardware [1, 2]. In our “Interactive Technical Illustration” paper [4], we chose to use environment maps because they provide the most flexibility in the shading model. This effectively allows us to evaluate a lighting model at every normal/reflection direction in the visible hemisphere in eye-space.

Shading is rendered using one of three modes. The first two are the diffuse and metallic shading models [2] presented in Section 8. In its simplest form the diffuse cool to warm shading model interpolates from a cool (blue-green) to a warm (yellow-orange) color based on the surface normal. This cool-to-warm diffuse shading is shown in Figure 6(a). The third method is an adaptation of this cool to warm shading, simulating the more dramatic shading effects sometimes used by artists. Figure 6(b) illustrates the effect achieved when the reflected light from the left of the object produces a back-splash of light opposite the



(a) Shading by Gooch et al.

(b) Shading with splash back

Figure 6: The dark banding in the light splash back model can communicate more curvature information and works well on organic models. (See Color Plate)

direct lighting source. This is accomplished by modifying the model of [2] with a simple multiplier:

$$(\alpha |\cos \theta| + (1 - \alpha))^p,$$

where α and p are free parameters which, for this image, are set to 0.76 and 0.78, respectively.

We evaluated the whole shading equation in a Phong environment map. In using an environment map as shown in Figure 5, all normals in eye-space are mapped to a 2D texture. This shading only is valid in eye-space, but it is possible to extend these to view-independent environment maps [6]. The cool-to-warm and light “splashback” terms are a function of the light direction and could be implemented with this representation. However, the Phong term would have to be computed for each view even though a single light source could be implemented as a single 1D texture map instead of a full 2D texture map.

10.6.1 Metal Shading

The metal shading technique we use assumes a principle direction of curvature and striping occurs in an orthogonal direction. We first compute a table of random intensities where sample is: $b + (r * a)$, where the base b is -0.1, r is a random number in [0,1] and a is 1.4. This causes the distribution be to biased towards white and black. We then filter each element in the table with each of its neighbors using a 1-5-1 weighting scheme and clamp this value to be in the range of [0,1]. We make these values periodic so there is some coherence which will remain smooth as they wrap around the model.

The table is then resampled into a 1D texture map. The texture map is used as a cosine distribution because it is indexed via a dot product. The resampling makes sure the bands are uniformly distributed on a cylinder. We then render the model with this texture map. The texture matrix computes the dot product with a fixed axis orthogonal to the principle curvature direction, and remap the value into [0,1]. This technique can be scaled in order to change the spacing of the stripes.

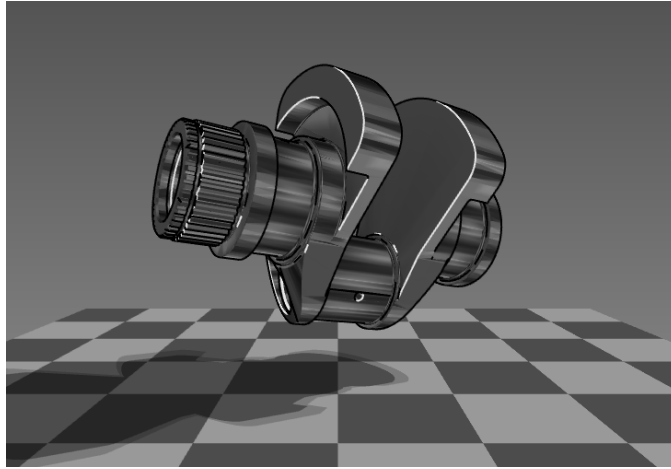


Figure 7: Metal-shaded object with shadow and ground plane. White creases and black silhouette lines are also drawn.

By itself, this texture does not look convincing, therefore we add Phong highlights computed by lighting a texture map in eye space with several Phong light sources oriented in the directions of a icosahedron's vertices, shown in Figure 7. A fairly large specular power, empirically around 30-50, seemed to work best with a specular coefficient of about 0.3.

10.6.2 Shadowing

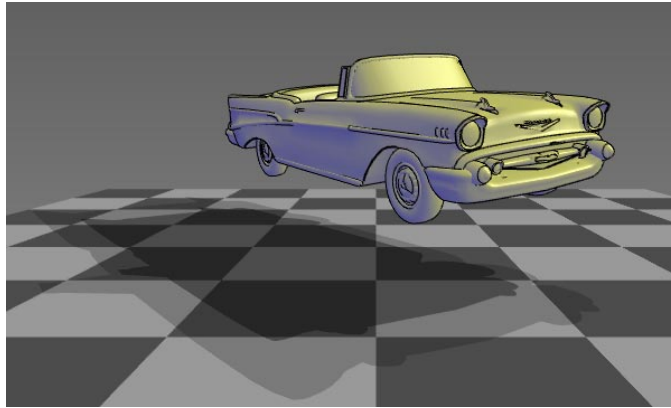
We draw shadows in one of three modes: a shadow with a hard umbra and a hard penumbra, a single hard shadow, and a soft shadow, as shown in Figure 8. Both of the later two modes approximate a spherical light source at a fixed distance from the center of the model in the direction of the light source used for shading.

The simplest and fastest method to draw simple shadows is to explicitly draw an umbra and penumbra. We draw two hard shadows, one from the center of the spherical light source back in the direction used for shading, and the other forward.

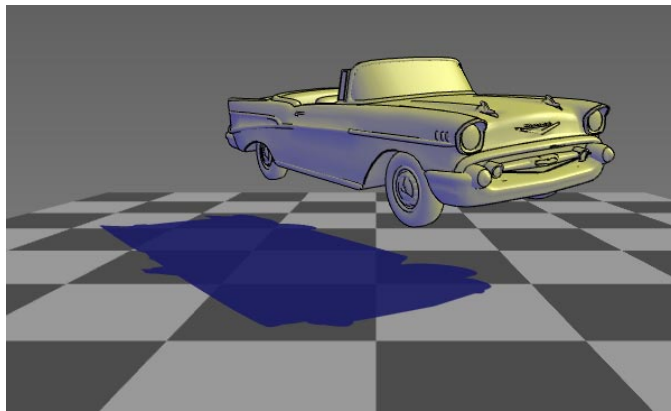
Soft shadows are problematic to render both accurately and efficiently, so we use an approximation to gain speed. Instead of using the conventional method to simulate an area light source, i.e., sampling the area light source and accumulating the point approximations, we project multiple shadows from the center of the approximation sampling a 1D direction, the ground plane's normal. This is done by projecting the same shadow onto a stack of planes, then translating the shadows to the ground plane and accumulating them, as shown in Figure 9.

Note that with this method, each "sample" is a perspective remapping of the first, intersected on a different plane. We could render a single shadow, copy it into texture memory and then remap it correctly to accumulate the other samples. This is much faster than projecting multiple jittered samples since there is a lower depth complexity for rasterization and a much lower burden on the transformation if the texture mapping method were used.

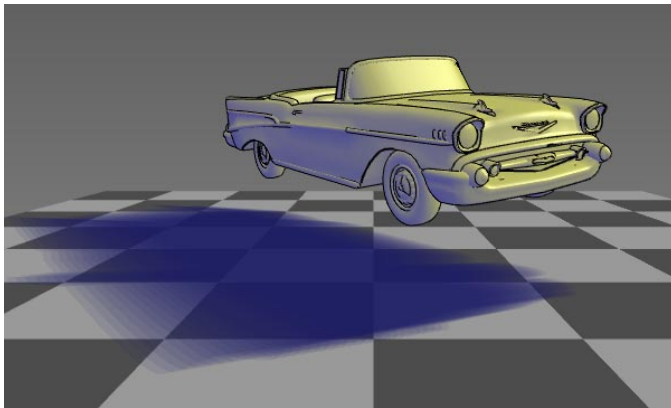
This method assumes that the silhouette from different points on the spherical light source is the same, i.e., the projection is the same. The planes coming out of the receiver will not correctly model contact. However, you can render only the lower planes if contact occurs resulting in a less realistic shadow, but one without distracting spillover.



(a) Hard penumbra and hard umbra.



(b) Single hard, colored shadow.



(c) Colored soft shadow.

Figure 8: Shadows provide valuable information about three-dimensional structure, especially the spatial layout of the scene. (See Color Plate)

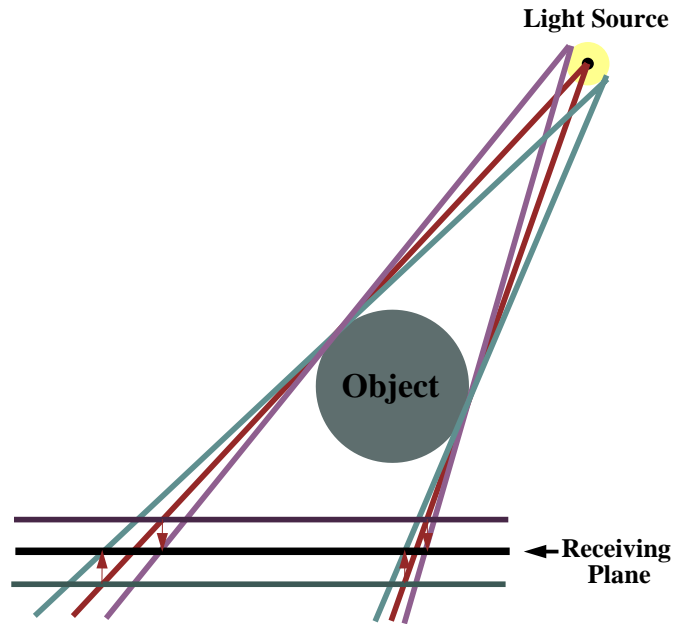


Figure 9: Drawing the shadow of a sphere with a spherical light source directly onto a ground plane below, traditionally each sample will render an ellipse. To get an accurate representation of the penumbra, this surface of the spherical light source needs to be sampled in 2 dimensions. With our method, each shadow is a concentric circle, requiring less samples to get the same results.

10.7 Conclusion

One of the largest goals in computer graphics has been realistic image synthesis. However, in a number of situations, an image which highlights particular information is valued above realism. For example, an automobile repair manual uses illustrations to remove unnecessary details and to draw attention to specific features.

Many computer-generated images have to be hand-tuned and they still convey shape poorly. The goal of this research was to use the techniques explored by illustrators for centuries to automatically generate images like technical illustrations and to be able to interact with these illustrations in three dimensions.

Acknowledgment

This work was done in collaboration with Peter-Pike Sloan, Peter Shirley, Elaine Cohen, and Rich Riesenfeld.

Authors' Note

These notes should be read while looking at colored images. See the course notes on the SIGGRAPH 99 CD-ROM for the images if colored images do not accompany this document.

References

- [1] David Blythe, Brad Grantham, Scott Nelson, and Tom McReynolds. *Advanced Graphics Programming Techniques Using OpenGL*. http://www.sgi.com/Technology/OpenGL/advanced_sig98.html, 1998.

-
- [2] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A Non-photorealistic Lighting Model for Automatic Technical Illustration. In *Computer Graphics*, July 1998. ACM Siggraph '98 Conference Proceedings.
 - [3] Amy A. Gooch. Interactive non-photorealistic technical illustration. Master's thesis, University of Utah, December 1998.
 - [4] Bruce Gooch, Peter-Pike Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. *Interactive 3D Conference Proceedings*, April 1999.
 - [5] Wolfgang Heidrich. A model for anisotropic reflections in open gl. In *SIGGRAPH 98 Conference Abstracts and Applications*, page 267, July 1998.
 - [6] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 39–45, September 1998.
 - [7] Michael A. Kowalski and Lee Markosian et al. Art-based rendering of fur, grass, and trees. In *SIGGRAPH 99 Conference Proceedings*, August 1999.
 - [8] L. Markosian, M. Kowalski, S. Trychin, and J. Hughes. Real-Time Non-Photorealistic Rendering. In *SIGGRAPH 97 Conference Proceedings*, August 1997.
 - [9] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 1993.
 - [10] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. *Symposium on Interactive 3D Graphics*, April 1999.
 - [11] Bruce Walter, Gun Alpay, Eric P. F. Lafortune, Sebastian Fernandez, and Donald P. Greenberg. Fitting Virtual Lights for Non-Diffuse Walkthroughs. In *SIGGRAPH 97 Conference Proceedings*, pages 45–48, August 1997.
 - [12] H. Zhang and K. Hoff III. Fast backface culling using normal masks. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 103–106, April 1997.