# Interactive Technical Illustration

**Bruce Gooch** **Peter-Pike J. Sloan** **Amy Gooch** **Peter Shirley** **Richard Riesenfeld**

Department of Computer Science
University of Utah
http://www.cs.utah.edu/

## Abstract

A rendering is an abstraction that favors, preserves, or even emphasizes some qualities while sacrificing, suppressing, or omitting other characteristics that are not the focus of attention. Most computer graphics rendering activities have been concerned with photorealism, i.e., trying to emulate an image that looks like a high-quality photograph. This laudable goal is useful and appropriate in many applications, but not in technical illustration where elucidation of structure and technical information is the preeminent motivation. This calls for a different kind of abstraction in which technical communication is central, but art and appearance are still essential instruments toward this end. 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. A paradigm for the display of technical illustrations in a dynamic environment is presented. 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 includes 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. This paper, all of the images, and a mpeg video clip are available at *http://www.cs.utah.edu/∼bgooch/ITI/*.

**CR Categories:** I.3.0 [Computer Graphics]: General; I.3.6 [Computer Graphics]: Methodology and Techniques.

**Keywords:** interaction, illustration, non-photorealistic rendering, silhouettes, lighting models, material properties, hardware rendering.

## 1 Introduction

Currently the process of documenting computer-aided design projects is undergoing vast changes. Systems are being developed to automatically create user and repair manuals during the design phase of a project. In order to document an entire manufactured object, six or more static images may be needed to show top, bottom, left, right, front, and back sides of the object. These images would need to be redone each time a new part or procedure was documented. Recently online documentation, as well as online shopping networks for mechanical parts [11], have been introduced. Instead of a series of static technical illustrations, more information could be provided if users could interact with a 3D model of the part being documented or sold. 3D viewers are for the most part based on traditional computer graphics lighting and shading models. It has been shown that traditional hand-drawn technical illustrations do a better job of describing the shape, structure and material composition of

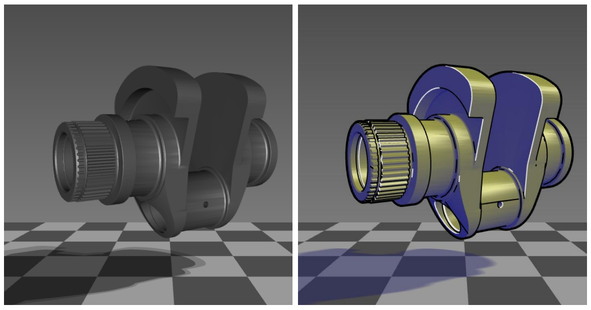


Figure 1: Left: Phong-shaded model. Right: Cool to warm shading, including silhouettes and creases as used by technical illustrators.

objects than traditional computer graphics [6]. The central idea of this paper is to extend the techniques of static technical illustration to interactive three dimensional display.

In Betty Edwards' book, *Drawing on the right side of the brain* [4], she lists five perceptual skills that an artist needs to have in order to produce quality drawings: 1 the perception of edges, 2 the perception of spaces, the 3 perception of relationships, the 4 perception of light and shadow, and the perception 5 of the whole or gestalt. Our work attempts to aid in the communication of shape information by enhancing the displayed model, catering to these perceptual skills in the user.

We have extended a number of two dimensional techniques into a three dimensional viewing package. These techniques include line weight depth cuing, shading and shadowing, light and highlight motion, and a non-parametric method of representing metal. The combination of these effects from traditional illustration and the impact they have on the human visual system have allowed us to create a system for viewing models in 3D which communicate far better than traditional rendering algorithms would allow. In Section 2 we review previous computer graphics work and conclude that little has be done to create interactive technical illustrations. In Section 3 we analyze the conventions for producing static illustrations and then discuss the issues involved in creating interactive illustrations in Section 4. Finally, in Section 5 we present the implementation details both in software and using high-end computer graphics hardware.

## 2 Related Work

Computer graphics algorithms that imitate non-photographic techniques such as painting or pen-and-ink are referred to as non-photorealistic rendering (NPR). An underlying assumption in NPR is that artistic techniques developed by human artists have intrinsic merit based on the evolutionary nature of art. For this reason techniques are usually borrowed directly from artists rather than reinvented from first principles.

Non-photorealistic techniques used in computer graphics [3, 5,

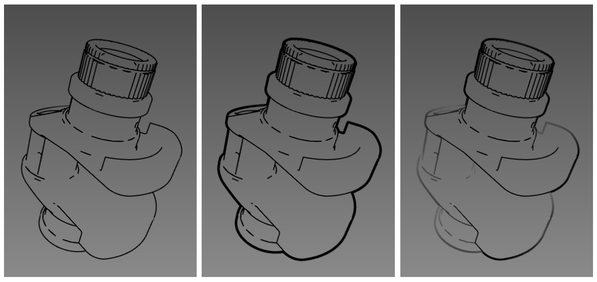


Figure 2: Three line conventions suggested by Martin [14]. Left: single weight used throughout the image. Middle: heavy line weight used for outer edges, other lines are thinner. Right: vary line weight to emphasize perspective.

6, 7, 12, 13, 16, 22, 26] vary greatly in their level of abstraction. Those that produce a loss of detail, like watercolor or pen-and-ink, produce a high level of abstraction. However, several previous papers use a low level of abstraction which preserves precise shape properties and are thus well suited to technical illustration, but lack interaction.

Although there is a wealth of computer graphics research dealing with the display of three dimensional images, there has been little exploration into utilizing artistic techniques to aid in the conveyance of shape information in an interactive setting. Markosian et al. [13] developed algorithms for probabilistically calculating only the silhouettes for polyhedral models in real time. There are also 3D paint programs which allow the user to experiment with non-photorealistic methods [19, 23] but these methods restrict interaction and require users trained in traditional drawing and painting techniques. In contrast, our methods incorporate user interaction with *automatically* generated 3D technical illustrations based on geometric models.

## 3 Static Illustration Principles

Human-drawn technical illustrations are usually stand-alone images from a single viewpoint presented on a non-stereo medium such as pen on paper. In this section we discuss the components of such illustrations that we use in a computer graphics context: line character, shading and shadowing.

### 3.1 Lines in Technical Illustration

Previous foundational research [5, 6, 13, 21] established which lines should be drawn in a computer generated image to maximize the amount of information conveyed while minimizing the number of lines drawn. They observed that illustrators use edge lines, consisting of surface boundaries, silhouettes, discontinuities, and creases to separate individual parts and to suggest important features in the shape of each object. These static images represented edge lines with black lines of uniform weight.

Varying type of lines

There are many line weight conventions which the illustrator chooses among based on the intent of the image. Martin [14] discusses three common conventions shown in Figure 2: a single line weight used throughout the image, two line weights with the heavier describing the outer edges, and varying the line weight along a single line emphasizing the perspective of the drawing with heavy lines in the foreground. One way of achieving this effect in raster graphics is to vary the line weight dependent upon the direction of the light source or in an user specified direction, giving a shadowed effect. However, most illustrators use bold external lines, with thinner interior lines, which aid in the perception of spaces [4].

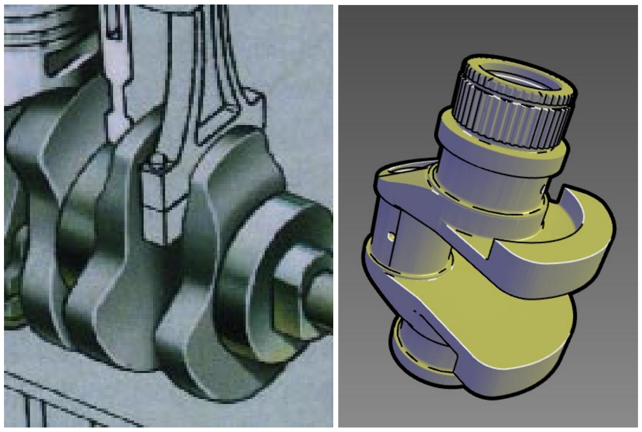


Figure 3: Left: Illustrators sometimes use the convention of white interior edge lines to produce a highlight. Image copyright 1995 Macmillan [20]. Used by permission. Right: An image produced by our system, including shading, silhouettes and white crease lines.

edge lines:black
interior lines:white

In almost all illustrations, edge lines are drawn in black. Occasionally, if the illustration incorporates shading, another convention is used in which some of the interior lines are drawn in white, like a highlight. Lines drawn in black and white suggest a light source and denote the models orientation. For example, Figure 3 compares an illustration produced by an artist and an image from our system in which white creases are drawn.

### 3.2 Shading

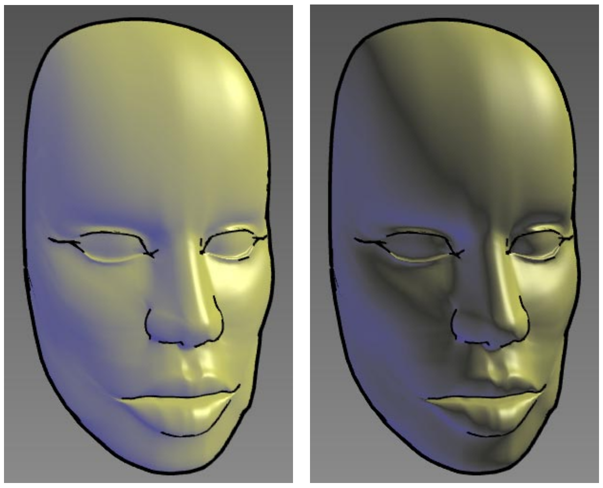
For shading we use one of three modes. The first two are the diffuse and metallic shading presented by Gooch et al. [6]. In its simpliest form the cool to warm shading interpolates from a cool (blue-green) to a warm (yellow-orange) color based on the surface normal. This cool-to-warm diffuse shading is shown in Figure 4a. The third method is an adaptation of this cool to warm shading, simulating the more dramatic shading effects sometimes used by artists. Figure 4b illustrates the effect achieved when the reflected light from the left of the object produces a back-splash of light opposite the direct lighting source. This is accomplished by modifying the model of [6] with a simple multiplier:

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

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

### 3.3 Shadowing

Illustrators only include shadows when they do not occlude detail in other parts of the object [14, 15, 20]. In 3D interactive illustrations, adding only a drop shadow on a ground plane, not the shadows that an object may cast onto itself, provide helpful visual clues without occluding important details on the object. It is probably not important that these shadows be highly accurate to provide valuable information about three-dimensional structure, especially the spatial layout of a scene [10, 25]. We provide the option to display one of three types of shadow which will be discussed in Section 5.3, as well as the option to make the shadow colored, as done by many artists [18].



(a) Shading by Gooch et al. (b) Shading with splash back

Figure 4: The dark banding in the light splash back model can communicate more curvature information and works well on organic models.

## 4 Dynamic Illustration Principles

The question remains, how do the 2D illustration rules change for an interactive 3D technical illustration? Adapting the shading and line conventions presented earlier is fairly straightforward as long as the line weight 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 relevant protocols in traditional illustration, we may want to base these 3D illustration conventions on how one would move real objects. This has an effect on how the light changes with respect to the object, the light position can be specified as relative to the object or to the viewer.

### 4.1 Viewer Versus Object Motion

The shading models presented in Section 3.2 are used to full advantage if the surface color varies completely from cool to warm. This involves moving the object and not the viewpoint or the lights. As seen in Figure 5, moving the object while holding the camera and light positions constant presents more shape information and surface detail. For this reason our interface rotates the object rather than the viewer, leaving the background, light, and viewer in place.

When multiple objects appear in a scene, illustrators often use different shading across each object, inferring that each object has its own light, which does not affect other objects in the environment, similar to the virtual lights described by Walter et al. [24]. For example, two objects in a scene may be lit differently to draw attention to different attributes of each object. If this were accomplished by adding two lights to the environment, the multiple highlights would be confusing.

### 4.2 Material Properties

Most material properties are nearly constant as the view direction or lighting changes. However, the metal shading presented by Gooch et al. is the replication of the anisotropic reflection [8] due to the surface of the object and the reflection of the environment. When a real metal part is rotated in one's 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 change abruptly. Using a metal texture would be more appropriate and a metal texture in an interactive environment would still properly convey the material property.

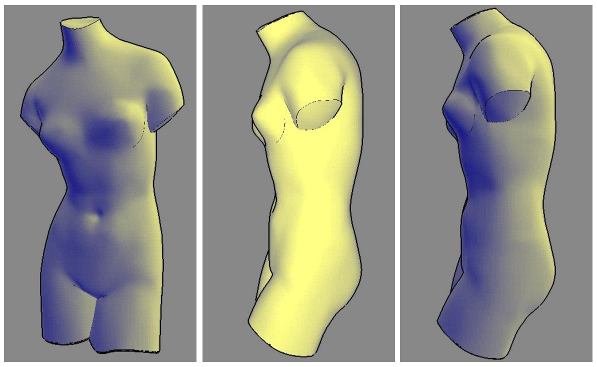


Figure 5: Left: Model with cool to warm shading with lights positioned up and to the right. Middle: After the camera position is moved to view the side of the model. Right: After moving the object instead of the camera, allowing the surface to vary completely from cool to warm.

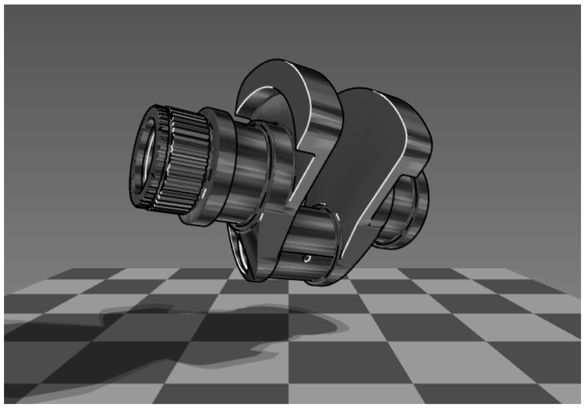


Figure 6: Metal-shaded object with shadow and ground plane. White creases and black silhouette lines are also drawn.

## 5 Implementation

As outlined in the previous two sections, our system needs the capability to interactively display a custom shading model, silhouettes, and interior edges. In addition, this interaction must be possible for complex geometric models. In this section we describe a variety of techniques for achieving these goals, and describe the tradeoffs involved in choosing a particular technique.

### 5.1 Displaying Important Edges and Silhouettes

To draw silhouettes, we have implemented several methods of extracting and displaying edge lines from polyhedral models, which will be discussed in Section 5.1.1 and 5.1.2. They 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

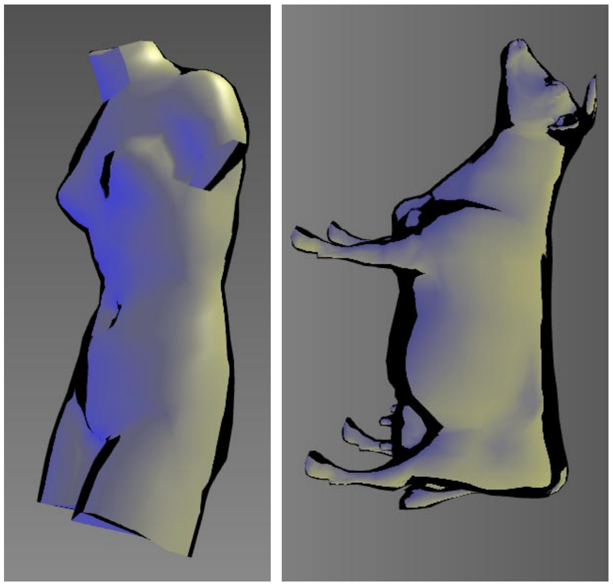


Figure 7: Adding the silhouettes to the environment map instead of calculating silhouettes from the geometry produces interesting artistic effects.

and software algorithms clearly have a place. The set of hardware methods are useful because of ease of implementation. The software methods are advantageous 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 5.1.3.

### 5.1.1 Hardware Methods

Using multi-pass rendering [1] there are several ways to extract silhouettes. The algorithm presented in the SIGGRAPH 1998 OpenGL Course doesn't capture internal silhouette edges and requires four passes of rendering. We recently found out that there is concurrent work similar in spirit to our hardware methods [2]. 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) [17] needs to be used. This function effectively modifies the depth values of the first pass based on

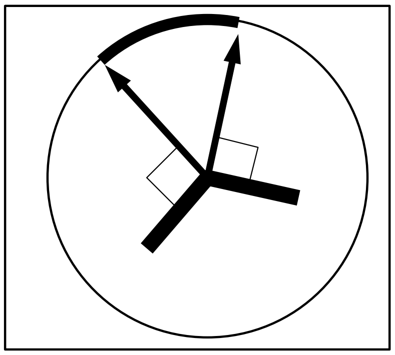


Figure 8: The arc in a Gauss map seen in 2D. The two bold line segments are faces that share a vertex. The orientations of their normals can be represented as points on the circle. The arc between those two points represents all orientations swept out between the two normals. In 3D the same reasoning applies, and the arc is an arbitrary segment on a great circle.

the slope of the triangles and a bias factor. This technique can be used to 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 7(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 7(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.

### 5.1.2 Software Methods

A straightforward technique for drawing 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. [27]. 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, 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. [27], 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 8). Under orthographic projection, a plane through the origin in this sphere defines the view. 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

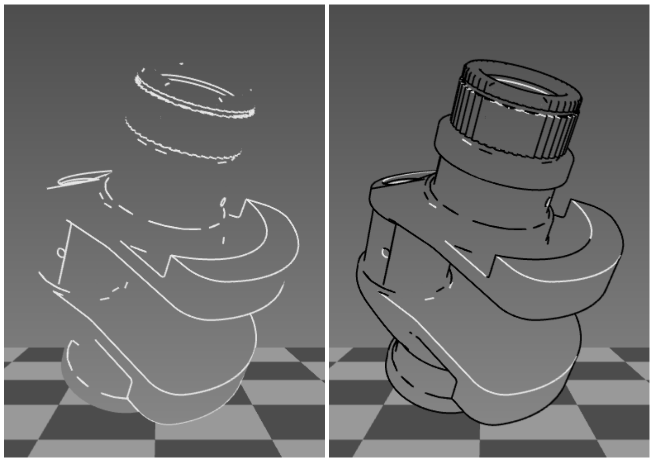


Figure 9: All creases are drawn in white (Left), and then all of the silhouette lines are drawn in black (Right), overlapping the creases.

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. This makes silhouette extraction logarithmic in the number of edges for smooth models where the arcs tend to be short. An arc is stored at the lowest possible level of the hierarchy. 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 [27] would minimize this effect. One advantage of any software approach is that it makes it easier to implement different styles of line drawing.

### 5.1.3 Line Styles

As discussed in Section 3.1, 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. Using a 1D texture map, there can be a relationship between the surface and a distance to a light or to a plane 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 require only one extra pass, and that pass may only be of the silhouette edges.

| 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.

| 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.

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 9.

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.

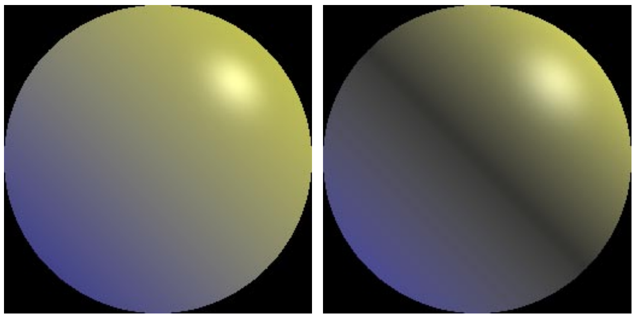
### 5.1.4 Discussion

Silhouette finding using specialized graphics hardware 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 [27] 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 4 levels of subdivision, i.e., 1280 faces. On the sphere this method is extremely efficient. When using overlapping 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 0. 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



(a) Environment map used to generate Figure 4(a).

(b) Environment map used to generate Figure 4(b).

Figure 10: Shaded sphere images used for environment maps.

takes only three-thousands 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 effective.

## 5.2 Shading

There are several ways to apply NPR shading models using hardware [6, 1]. 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.

We evaluated the whole shading equation in a Phong environment map. In using an environment map as shown in Figure 10, 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 [9]. The cool-to-warm and light "splashback" terms mentioned in Section 3.2 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.

### 5.2.1 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 it to be in the range of [0,1]. We make it periodic so there is some coherence which will remain smooth as it wraps 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.

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. A fairly large specular power, empirically

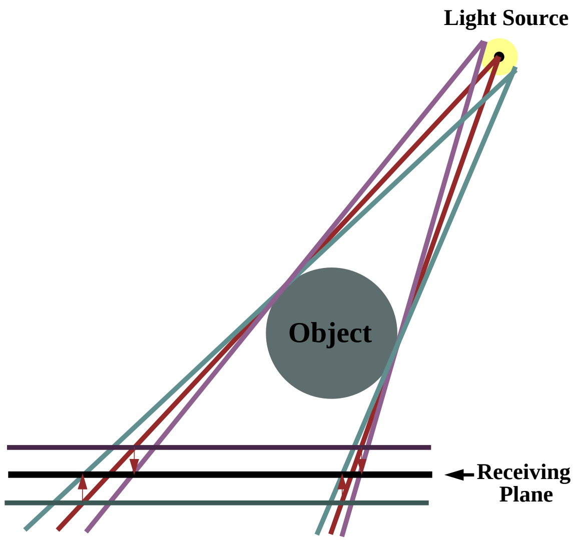


Figure 11: Drawing the shadow of a sphere with a spherical light source directly onto a ground plane directly below it, traditionally each sample will render an ellipse. To get an accurate representation of the penumbra, this surface of the spherical light source needs to be sampled in 2 dimensions. With our method, each shadow is a concentric circle, requiring less samples to get the same results.

around 30-50, seemed to work best with a specular coefficient of about 0.3.

## 5.3 Shadowing

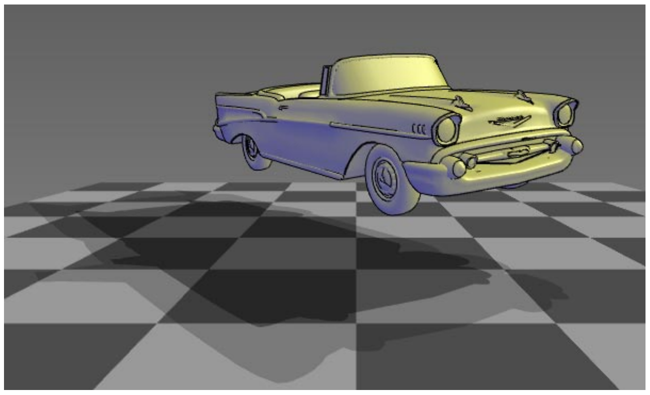
We draw shadows in one of three modes: a single hard shadow, a shadow with a hard umbra and a hard penumbra, and a soft shadow, as shown in Figure 12. Both of the later two modes approximate a spherical light source at a fixed distance from the center of the model in the direction of the light source used for shading.

The easiest 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.

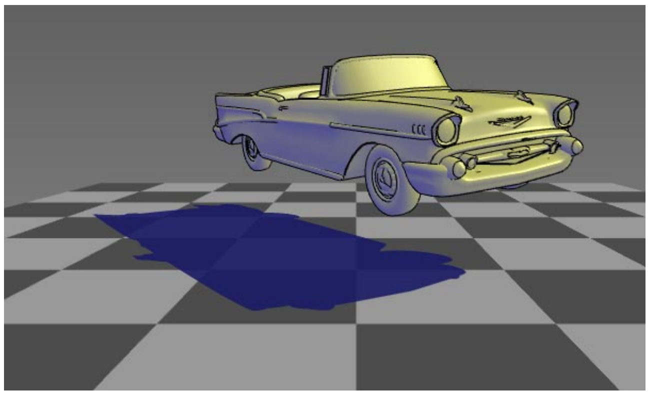
Soft shadows are problematic to do both accurately and efficiently, so we use an approximation to gain speed. Instead of using the conventional method to simulate an area light source, i.e., sampling the area light source and accumulating the point approximations, we project multiple shadows from the 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 11.

One thing to note is 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.

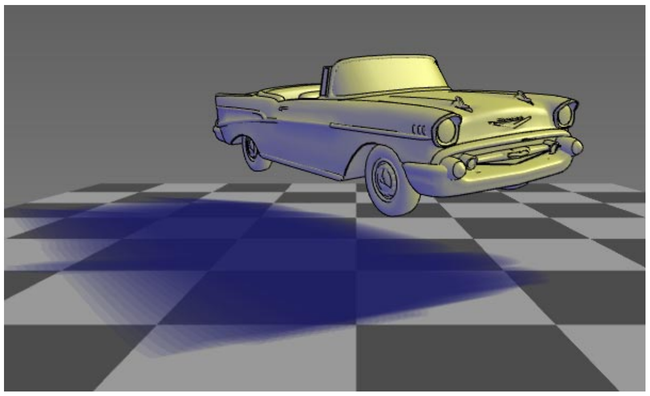
This method assumes that the silhouette from different points on the spherical light source is the same, i.e., the projection is the same. The planes coming out of the receiver will not correctly model contact. However, you can render only the lower planes if contact occurs resulting in a less realistic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



(a) Hard penumbra and hard umbra.



(b) Single hard, colored shadow.



(c) Colored soft shadow.

Figure 12: Shadows provide valuable information about three-dimensional structure, especially the spatial layout of the scene.

## 6 Future Work and Conclusion

We have reported on an approach that produces interactive technical illustrations. This work incorporates established principles from traditional art in a framework of powerful geometric modeling and rendering. The goal is to produce non-photorealisitic images for technical illustration, as well develop the algorithms which exploit aspects of the standard graphics pipeline to achieve interactive speeds. Although this system is formed by empirical data on human perception, a number of parameter settings are available to the user based on their aesthetic preferences. Our approach incorporates: parameters for cool and warm color choices, new interactive silhouette algorithms, various shadow schemes, line choices for internal features versus external outlines, and rendering methods exploiting texture maps. Together these produce technically informative and aesthetically pleasing interactive illustrations.

This paper addresses a widely occurring need, namely, generating attractive and informative technical illustrations for modern documentation schemes, which are likely to be hierarchical and web-based for individual exploration. It represents an advance in the relatively under-developed area of computer graphics involving non-photorealistic rendering, where the needs are extensive and the available methods are few. Inspired by the work of Markosian et al.'s real-time probabilistic silhouette finding methods, we have taken the next step to create a system which incorporates fast deterministic silhouette and crease finding algorithms, with artistic shading and shadowing.

These interactive illustrations emphasize the structure and detail of mechanical models. We believe that in the future, this approach can also be tuned for many other important domains such as medical illustration. We would also like to further explore changing line width along a single silhouette, calculating soft shadows from silhouettes, as well as creating new algorithms aimed at controlling silhouette frame-to-frame coherence.

## Acknowledgments

Thanks to Richard Coffey, Gordon Kindlmann, the members of the University of Utah Computer Graphics groups for help in the initial stages of the paper, and to Ramesh Raskar and Michael Cohen for discussing their work in progress with us. This work was supported in part by DARPA (F33615-96-C-5621) and the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

## 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] Michael Cohen and Ramesh Raskar. Personal Communication. 1998.

[3] Cassidy J. Curtis, Sean E. Anderson, Kurt W. Fleischer, and David H. Salesin. Computer-Generated Watercolor. In *SIGGRAPH 97 Conference Proceedings*, August 1997.

[4] Betty Edwards. *Drawing on the Right Side of the Brain*. Jeremy P. Tarcher/Putnam, 1989.

[5] Gershon Elber and Elaine Cohen. Hidden Curve Removal for Free-Form Surfaces. In *SIGGRAPH 90 Conference Proceedings*, August 1990.

[6] 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.

[7] Paul Haeberli. Paint By Numbers: Abstract Image Representation. In *SIGGRAPH 90 Conference Proceedings*, August 1990.

[8] Wolfgang Heidrich. A model for anisotropic reflections in open gl. In *SIGGRAPH 98 Conference Abstracts and Applications*, page 267, July 1998.

[9] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 39–45, September 1998.

[10] D. Kersten, D. C. Knill, P. Mamassian, and I. Bulthoff. Illusory motion from shadows. *IEEE Computer Graphics and Applications*, 379(31), 1996.

[11] ViewPoint Data Labs. *LiveArt 98*. Orem, UT, 1998.

[12] Peter Litwinowicz. Processing Images and Video for an Impressionistic Effect. In *SIGGRAPH 97 Conference Proceedings*, August 1997.

[13] L. Markosian, M. Kowalski, S. Trychin, and J. Hughes. Real-Time Non-Photorealistic Rendering. In *SIGGRAPH 97 Conference Proceedings*, August 1997.

[14] Judy Martin. *Technical Illustration: Materials, Methods, and Techniques*, volume 1. Macdonald and Co Publishers, 1989.

[15] Scott McCloud. *Understanding Comics*. Tundra Publishing Ltd., Northhampton, MA, 1993.

[16] Barbara J. Meier. Painterly Rendering for Animation. In *SIGGRAPH 96 Conference Proceedings*, August 1996.

[17] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 1993.

[18] Jose M. Parramon. *The Book of Color*. Watson-Guptill Publications, New York, NY, 1993.

[19] PartNet. *http://www.partNet.com/*. 423 Wakara Way Suite 216 Salt Lake City, Utah 84108, 1998.

[20] Tom Ruppel, editor. *The Way Science Works*, volume 1. MacMillan, 1995.

[21] Takafumi Saito and Tokiichiro Takahashi. Comprehensible Rendering of 3D Shapes. In *SIGGRAPH 90 Conference Proceedings*, August 1990.

[22] Mike Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *SIGGRAPH 97 Conference Proceedings*, August 1997.

[23] Daniel Teece. 3d painting for non-photorealisitic rendering. In *SIGGRAPH 98 Conference Abstracts and Applications*, page 248, July 1998.

[24] Bruce Walter, Gun Alppay, 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.

[25] Leonard R. Wanger, James A. Ferwerda, and Donald P. Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, May 1992.

[26] Georges Winkenbach and David H. Salesin. Computer Generated Pen-and-Ink Illustration. In *SIGGRAPH 94 Conference Proceedings*, August 1994.

[27] 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.