

## **Silhouette Extraction**

Bruce Gooch, *University of Utah*

---

### ***Abstract***

In computer graphics, silhouette finding and rendering has a central role in a growing number of applications. The silhouette is the simplest form of line art and is used in cartoons, technical illustrations, architectural design and medical atlases. In non-photorealistic rendering (NPR), complex models and scenes are rendered as simple line drawings by rendering silhouette edges.

In addition to NPR, silhouettes are advantageous in realistic rendering and in interactive techniques. Sander et al. explain that high polygon count models can be rendered at interactive rates by clipping the polygons of a coarse geometric approximation of a model along the silhouette of the original model. Hertzmann and Zorin have shown that silhouettes can be used as an efficient means to calculate shadow volumes. Haines demonstrates an algorithm using silhouettes for rapidly creating and rendering soft shadows on a plane. Johnson and Cohen show that haptic rendering can be facilitated using silhouette information. Some authors,] have described the use of silhouettes in CAD/CAM applications. Systems have also been built which use silhouettes to aid in modeling and motion capture tasks.

The silhouette set of a model can either be computed in object space or in screen space. Object space algorithms involve computations in three dimensions and produce a list of silhouette edges or curves for a given viewpoint. Screen space algorithms usually involve image processing techniques and are useful if rendering silhouettes is the only goal. Additional important feature lines do exist for three dimensional models. These lines include; texture boundaries, creases, and object boundaries. These additional feature lines have the convenient property of being view independent, and can therefore be completely specified prior to runtime.

This lecture examines both object space and image space silhouette extraction algorithms. The algorithms are compared in terms of code complexity, necessary system resources, and run time performance on a variety of polygonal models. The goal of this lecture is to become an informative source for programmers making a choice between these algorithms and methods.

---

## *Outline*

### I. Object Space Silhouette Algorithms

- a) **Brute Force** -- Iterate through each edge in a polygonal model and test whether each edge is a silhouette edge.
- b) **Edge Buffer** -- Using the "Edge Buffer" data structure of Buchanan and Sousa to iterate over facets instead of edges.
- c) **Probabilistic** -- An edge tracing method where a finite number of "seed" edges are chosen of reach viewpoint based on a measure of the likelihood that the "seed" edges are silhouettes.
- d) **Gauss Map Arc Hierarchy** -- The angles of arcs between front and back facing polygons are stored in a tree structure.
- e) **Normal Cone Hierarchy** -- Polygon normals are grouped into cones and these cones are stored in a tree structure
- f) **Implicit Surfaces** -- A silhouette tracing method where points on the silhouette curve are found using "ray tests". The silhouette curve is then traced along the surface in the direction of the view plane projection of the gradient.
- g) **NURBS Surfaces** -- Silhouette curves on a model are found by first using a "marching cube" algorithm to find surface patches which the silhouette curve passes through. Silhouette curves are then interpolated from the entry and exit points of the patch.

### II. Image Space Silhouette Algorithms

- a) **Two Pass Methods** -- Back facing polygons are rendered first with either their depth decreased or their field of view narrowed. Front facing polygons are then rendered on top.
- b) **Environment Map** -- Silhouette lines are added to a shading environment map as a preprocess.
- c) **One Pass Method** -- During a preprocess phase two cube maps are created, one of surface normals and one of eye linear maps. At runtime per pixel dot products are computed yielding silhouettes.
- d) **Model Augmentation** -- A second polygonal model is created with degenerate quads, the quads have no height, positioned normal to the surface at each edge of the model. At runtime each edge is checked to see if it is a silhouette. If an edge is a silhouette the corresponding quad is made non-degenerate i.e. the quad is given height.
- e) **Depth Discontinuity Methods** -- Pixel depth is compared on a per pixel basis, if the depth difference between two pixels is above a user defined tolerance one of the pixels is colored black.

# Silhouette Algorithms

## Bruce Gooch and Mark Hartner and Nathan Beddes

### 1 Introduction and Background

The silhouette is the simplest form of line art and is used in cartoons, technical illustrations, architectural design and medical atlases[14]. In non-photorealistic rendering (NPR), complex models and scenes are rendered as simple line drawings by rendering silhouette edges. Lake et al. present interactive methods to emulate cartoons and pencil sketching[20]. Gooch et al. built a system to interactively display technical drawings[12]. Rheingans and Ebert and Lum and Ma have built a NPR volume visualization systems which use silhouettes to emphasize key data in volume renderings[22, 28].

In addition to NPR, silhouettes are advantageous in realistic rendering and in interactive techniques. Sander et al. explain that high polygon count models can be rendered at interactive rates by clipping the polygons of a coarse geometric approximation of a model along the silhouette of the original model[29]. Hertzmann and Zorin have shown that silhouettes can be used as an efficient means to calculate shadow volumes[14]. Haines demonstrates an algorithm using silhouettes for rapidly creating and rendering soft shadows on a plane[13]. Johnson and Cohen show that haptic rendering can be facilitated using silhouette information[18]. Some authors,[4, 17] have described the use of silhouettes in CAD/CAM applications. Systems have also been built which use silhouettes to aid in modeling and motion capture tasks[2, 10, 21].

The silhouette set of a model can either be computed in object space or in screen space. Object space algorithms involve computations in three dimensions and produce a list of silhouette edges or curves for a given viewpoint. Screen space algorithms usually involve image processing techniques and are useful if rendering silhouettes is the only goal.

### 2 Definition of a Silhouette

Given  $E(u, v)$  as the eye vector, a point on a surface  $\sigma(u, v)$  with surface normal  $N(u, v)$  is a silhouette point if  $E(u, v) \cdot N(u, v) = 0$ , that is, the angle between  $E(u, v)$  and  $N(u, v)$  is 90 degrees. This relationship is demonstrated in Figure 2. This definition includes internal silhouettes as well as the object's outline, or halo. It is important to note that the silhouette set of an object is view dependent, that is the edges of a model that are silhouettes change based on the point from which the object is viewed.

Additional important feature lines exist for three dimensional models. These lines include; texture boundaries, creases, and object boundaries. These additional feature curves are view independent, and can therefore be completely specified prior to runtime. In this work we evaluate only runtime silhouette extraction algorithms.

### 3 Silhouettes for Polygonal Models

The silhouette set for a polygonal model is defined to be all edges in the model which are shared by both a front-facing and a back-facing polygon, as illustrated in Figure 1.

For uniformity throughout this work we assume that polygon normals point outward from surfaces. This assumption yields the following:

if  $N \cdot E < 0$  then the polygon is front-facing

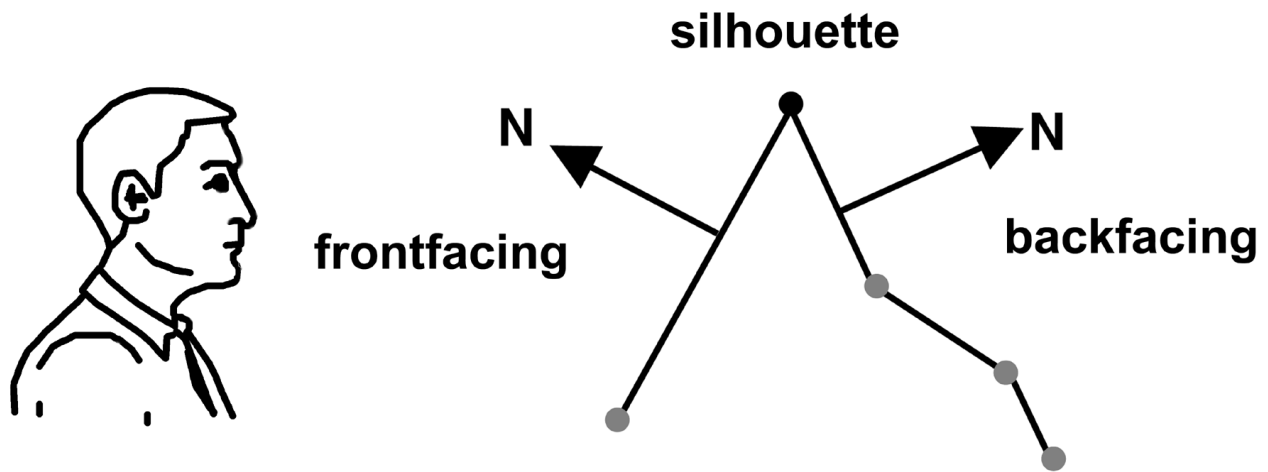


Figure 1: 2-D silhouette example for a polygonal surface.

if  $N \cdot E > 0$  then the polygon is back-facing

if  $N \cdot E = 0$  then the polygon is perpendicular to the view direction

## 4 Object Space Silhouette Algorithms

1. Brute Force – Iterate through each edge in a polygonal model and test whether each edge is a silhouette edge.
2. Edge Buffer – Using the “Edge Buffer” data structure of Buchanan and Sousa[3] to iterate over facets instead of edges.
3. Probabilistic – An edge tracing method where a finite number of “seed” edges are chosen of reach viewpoint based on a measure of the likelihood that the “seed” edges are silhouettes[23].
4. Gauss Map Arc Hierarchy – The angles of arcs between front and back facing polygons are stored in a tree structure[1, 12].
5. Normal Cone Hierarchy – Polygon normals are grouped into cones and these cones are stored in a tree structure[14, 18, 25, 29].
6. Implicit Surfaces – A silhouette tracing method where points on the silhouette curve are found using “ray tests”. The silhouette curve is then traced along the surface in the direction of the view plane projection of the gradient[6].
7. NURBS Surfaces – Silhouette curves on a model are found by first using a “marching cube” algorithm to find surface patches which the silhouette curve passes through. Silhouette curves are then interpolated from the entry and exit points of the patch[7, 11].

### 4.1 Brute Force

The brute force method of silhouette extraction requires testing each edge in the polygonal mesh sequentially to verify whether or not it is a silhouette. One needs to create a data structure which holds the information about an edge and contains pointers to the normal vectors of both polygons associated with that edge. Then create an edge list using a static array of these edge data structures. At runtime, for every frame, traverse the edge list. Each polygon

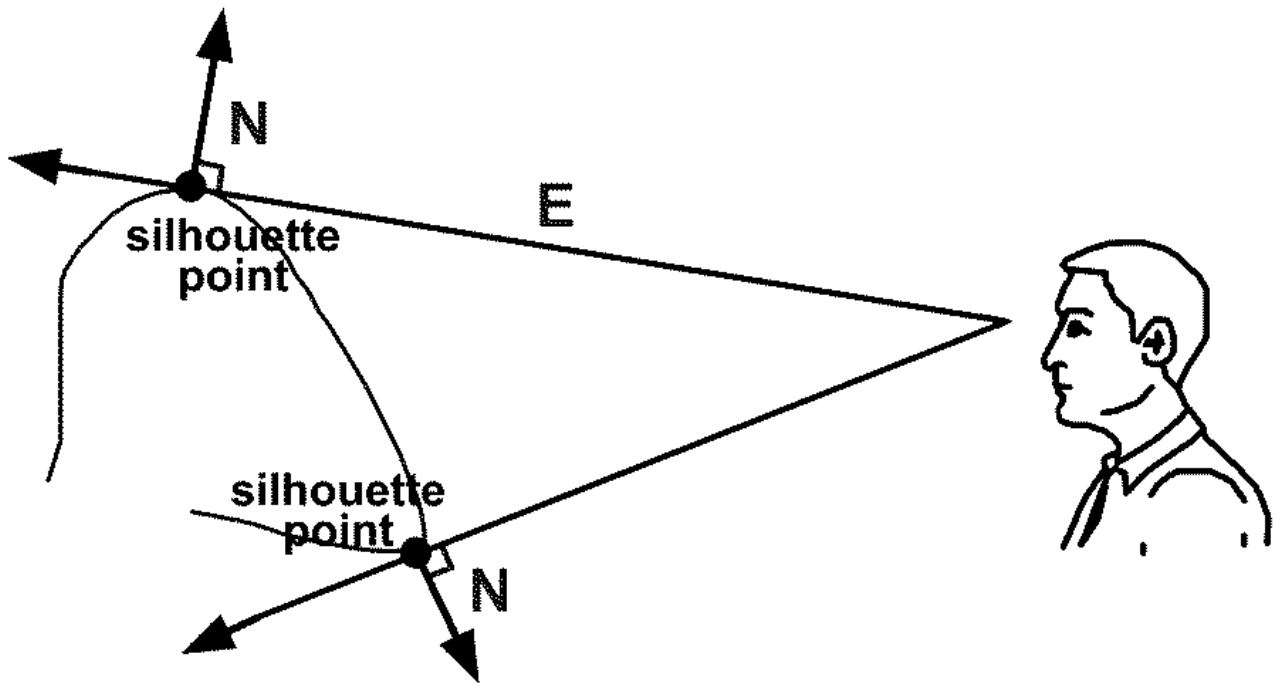


Figure 2: 2-D silhouette example for a smooth surface.

adjacent to the current edge is tested to determine whether it is front-facing or back-facing with respect to the current eye point. If one polygon is front-facing and the other back-facing a silhouette is rendered.

There is a steep dropoff initially due to cache performance, then the brute-force silhouette method's runtime complexity scales linearly with the number of edges. Because of the simplicity of the brute force method, it is easy to implement using a single iterative loop, thus eliminating any function call overhead.

## 4.2 Edge Buffer

Instead of iterating over each edge and testing both adjacent polygon normals for front/back facing, the Edge Buffer method iterates over the polygons [3]. Since the number of polygons is always fewer than the number of edges, the edge buffer method should run faster than the brute force algorithm. Create an edge list similar to the data structure used in the brute force method, adding of a front facing flag and a back facing flag for each edge. The front facing and back facing flags for each edge are initialized to 0. In addition, create a data structure which maps polygons back to edges shared by the respective polygon.

For each polygon in the polygon list, test to see whether the polygon is front-facing. If the polygon is front facing, XOR a 1 with the front facing flag for each edge shared by that polygon. Likewise, if the polygon is back facing, XOR a 1 with the back facing flag for each edge shared by that polygon. Upon completion, each edge that shares exactly one front-facing polygon and one back-facing polygon will have both flags set. Finally, iterate through the edge list and draw only the edges that have both their flags set and reset both of the flags for all the edges to 0.

In practice our implementation of the edge buffer algorithm actually runs slightly slower than brute force. Although the edge buffer method is less complex than brute force in terms of floating point operations, the edge buffer has a higher overhead cost because the XOR operations are being performed, in addition to the dot products computed to test whether polygons are front or back facing.

As outlined in the edge buffer paper [3], it is necessary to do a large number of edge table lookups at runtime. We created the polygon list during the pre-process stage to eliminate all table lookups during runtime. Without this optimization, the edge buffer runs much slower than brute force. The edge buffer's runtime routine is implemented

---

using a single loop, thus eliminating function call overhead.

### 4.3 Probabilistic

Markosian et al. [23] present a probabilistic silhouette finding algorithm. A small number of edges are chosen based on the probability that they are silhouette edges, then tested to see if they are silhouette edges. Edges with higher dihedral angles have a higher probability of being silhouettes. Silhouette edges from the previous frame also have a high probability of being silhouettes, or being close to the new silhouette edges. When a silhouette edge is found, its adjacent edges are tested to see if any of these edges is also a silhouette edge.

Compute the dihedral angle of each edge in a model, sort the edges by their dihedral angle, and place the sorted edges into a list. We call this list the dihedral angle list. The probability that an edge is a silhouette is  $(\theta/\pi)$  where  $\theta$  is the dihedral angle between the edges two adjacent polygons. At runtime choose random edges to test as silhouettes and weight the random search to look at edges with a high probability of being silhouettes. The data structure used for individual edges can be modified from the brute force data structure with the addition of pointers to all adjacent edges.

A collection of edges is chosen and tested to see if they are silhouette edges. All the silhouette edges from the previously rendered frame are tested first. Next, a number of randomly chosen edges from the dihedral angle list are tested to determine if they are silhouette edges. Each edge that is found to be a silhouette edge for the current viewpoint is then traced.

If an edge is a silhouette, trace through the edge adjacency pointers in the dihedral angle list to check if any adjacent edges are silhouette edges. Adjacent edges are recursively tested until there are no adjacent silhouette edges. In order to avoid testing silhouette edges that have already been found, a reference to each silhouette edge is hashed into a table. Every time a new silhouette edge is found it is checked for inclusion in this hash table. In order to avoid having to reset the hash table after each frame, frame numbers are stored in the hash table and compared to the current frame number.

We observed a speedup over the brute force algorithm, although a small number of silhouette edges may be missed each frame. We also observed that frame-to-frame coherence of silhouette edges works very well, but can become more difficult as the model becomes very large because silhouette curves tend to move across many polygons and are therefore may be missed by the tracing algorithm. For good performance scaling, the number of random edges chosen at the beginning of each frame must be proportional to the square root of the total number of edges.

### 4.4 Gauss Map Arc Hierarchy

A modified Gauss map can be used to calculate the silhouette edges of a polygonal model [1, 12]. The model is placed at the origin of a bounding sphere (Gauss map) and each edge of the model maps to an arc on the sphere. Silhouette edges are extracted from the Gauss map by intersecting the Gauss map with a plane. The intersecting plane is defined as passing through a point at the origin of the bounding sphere and perpendicular to the viewing vector. The arcs on the Gauss map which are intersected by the plane correspond to silhouette edges in the original model. Since the Gauss map only takes into consideration the viewing direction, and not the viewing distance, this technique will not work for perspective viewing.

Begin by mapping the model edges onto the Gauss map. Each edge in the model has two adjacent facets,  $F_1$  and  $F_2$  which have normals  $N_1$  and  $N_2$ . Model edges are mapped to the Gauss surface by placing  $N_1$  and  $N_2$  at the origin of the Gauss Map and sweeping  $N_1$  across the Gauss surface to  $N_2$ . For a Gauss sphere,  $N_1$  and  $N_2$  sweep out an arc on the surface of the sphere.

The Gauss map can be represented as a bounding cube. With the Gauss map represented as a cube, the Gauss map arcs become straight lines on one or more of the cube faces. Each face of our cube is divided into a 20 by 20 grid of buckets, and each edge maps to several of these buckets. After all edges have been mapped each bucket contains a list of zero or more edges.

---

To extract the set of silhouette edges for a given viewing direction a plane is used to intersect the Gauss map. The intersecting plane is defined by a point P at the origin of the Gauss map and a vector V which is the viewing direction. Since our Gauss map is represented as a grid of buckets, the intersecting plane corresponds to a list of buckets intersected in the Gauss map. Each bucket contains a possibly empty list of edges which are silhouette edges for the current viewing direction.

This algorithm is simple to implement when the Gauss sphere is approximated by a cube. However, the data structures used in this algorithm can become large depending on the bucket resolution, and this technique works only for orthogonal projection. However, the Gauss Map method is ideal when sufficient quantities of memory are available.

## 4.5 Normal Cone Hierarchy

There are several silhouette extraction methods based on hierarchal culling. Hertzmann et al. use dual surface intersections [14]. Pop et al. use a wedge hierarchy [25]. Sander et al. introduce the idea of using two open-ended normal cones to compute whether or not an edge is a silhouette edge [29].

Each of these algorithms requires descending a hierarchal data structure at runtime. We choose to implement the normal cone method of Sander et al. because their method requires only a single dot to be computed at each level of descent. The methods of Hertzmann et al. [14] and Pop et al. [25] interpolate between edges to find the exact zero crossing of the silhouettes and require more complex calculations.

Sander et al. [29] create a hierarchy of normal cones during a pre-process, which can be used at runtime to cull large numbers of edges which are not silhouettes. Any cones which intersect the current eye vector are discarded as not being silhouette edges. Cones that do not intersect the eye vector have to be analyzed. The cones are organized in a hierarchical search tree. If the polygons in a node can be determined to be all front-facing or all back-facing, the node can be discarded as not containing silhouettes.

The algorithm of Sander et al. uses weighting functions based on linear programming to determine which cones can be combined when forming the search tree. These weighting functions are computationally expensive and cause the pre-process algorithm to take anywhere from 30 minutes to 24 hours to load a polygonal model.

First, divide the edges into groups based on their dihedral angles. Next, divide the unit sphere into eight regions and build normal cones for each of these regions. Each dihedral angle group is now sorted by normal to form the next level of the hierarchy. Next sort each of the normal cones with respect to spatial proximity to build the next level. Recursively continue the cone normal and spatial proximity sort process until each cone contains a single edge.

During runtime, traverse the cone hierarchy and test cones to find if they include the current eye vector. This test can be done with two dot products:

The eye vector is inside the cone if

$$(\text{eyePoint} - \text{coneOrigin}) \cdot (\text{scaledConeNormal}) \geq 0 \quad \text{and} \quad ((\text{eyePoint} - \text{coneOrigin}) \cdot (\text{scaledConeNormal}))^2 \geq \|\text{eyePoint} - \text{coneOrigin}\|^2 \quad \text{where} \quad \text{scaledConeNormal} = \text{coneNormal} / \cos(\text{coneAngle})$$

If a cone contains the eye point, discard it and all its sub-cones. Otherwise traverse down the cone hierarchy until only edges are left. Each edge that is not culled at this point must be checked individually to determine whether or not it is a silhouette edge.

## 4.6 Object Space Conclusions

We found that for small models, under 10,000 polygons for our hardware, brute force silhouette extraction is easy to implement and runs nearly as fast much more complex methods. For more complex models it may be worth the time implementing more complex methods. If orthographic is all that is needed, then Gauss Maps are easiest to implement and also very fast. For large models with perspective, methods based upon hierarchal culling may be necessary, but are generally difficult to program.

---

We found that silhouette extraction methods are more sensitive to size than to complexity in the polygonal models. In all of the tests we performed model size dominated the runtime speed of the algorithms. We tested the algorithms on the complexity based test suite of Kettner and Welzl [19] we found no significant difference in the runtime of the algorithms on models of differing complexity but with similar polygon counts. This may be due to the fact that all of the models in the Kettner and Welzl test suite have less than 15,000 polygons. The complexity based model test suite is available online at: [www.cs.unc.edu/kettner/proj/obj3d/index.html](http://www.cs.unc.edu/kettner/proj/obj3d/index.html)

We would like to thank Aaron Hertzmann for sharing his code with us. We would also like to thank Peter Pike Sloan, Amy Gooch, Lee Markosian, Mario Costa Sousa, Gershon Elber and Adam Finkelsein for their time and consideration in talking with us and answering our many questions on this project. We would also like to thank Amy Gooch for her help in constructing models.

Thanks to Alias|Wavefront for their generous donation of Maya Complete, which we used to create the geometric models for this paper.

## 5 Image Space Silhouette Algorithms

1. Two Pass Methods – Back facing polygons are rendered first with either their depth decreased or their field of view narrowed. Front facing polygons are then rendered on top[12, 27].
2. Environment Map – Silhouette lines are added to a shading environment map as a preprocess[12].
3. One Pass Method – During a preprocess phase two cube maps are created, one of surface normals and one of eye linear maps. At runtime per pixel dot products are computed yielding silhouettes[8].
4. Model Augmentation – A second polygonal model is created with degenerate quads, the quads have no height, positioned normal to the surface at each edge of the model. At runtime each edge is checked to see if it is a silhouette. If an edge is a silhouette the corresponding quad is made non-degenerate i.e. the quad is given height[15, 26].
5. Depth Discontinuity Methods – Pixel depth is compared on a per pixel basis, if the depth difference between two pixels is above a user defined tolerance one of the pixels is colored black[5, 16].

## 6 Two Pass Methods

There is no preprocessing stage.

During runtime the model is rendered in two passes. First the front facing polygons are drawn, moved slightly away. Then back facing polygons are drawn as thick lines, moved slightly closer to the viewer.

Time to write and debug the code was very short, this is trivial to implement. (11 lines of code)

The visual quality for this method is generally the best of the four.

The speed is fast and the method scales well.

No programmable hardware or special extensions are needed.

No extra memory is needed outside the normal amount needed for the models.

## 7 Environment Map

There is no preprocessing stage.

During runtime the model is rendered with a cube map that is indexed via face normals. The cube map is white with a uniform black line around the center, perpendicular to the viewer. Triangle vertices that have normals pointing to the black line are close to the silhouette points and are colored black, the others are colored white.

---

Time to write and debug the code was short. Because it uses extension that may not be covered in an undergraduate course it may take a little longer than the method above. 20 lines of code to set up the cube map, 2 lines in the display loop to turn the cube map on and off.

The visual quality of this method is generally below average, the silhouette line is wavy due to interpolating vertex colors across triangle interiors.

The speed is the fastest of the four.

In our implementation, we use several ARB extensions (which should be on any modern card at this point) but no programmable hardware.

In addition to the models, just a small amount of space is needed for the cube map which can be made ahead of time in a paint program.

## 8 One Pass Method

There is no preprocessing stage.

During runtime register combiners are used to take the dot product between the eye vector and the surfaces normal vector (both found in one of two cube maps) The alpha values of the normal map are scaled in its mipmap, scaling the dot product (and therefore the color).

Time to write and debug the code was medium to long. Probably lower with pixel shader parsers we did not have access to at the time. 9 lines in the display loop to turn the cubemaps on and off and to turn the register combiners on and off. 50 lines of code to set up cube maps and combiners.

The visual quality of this method is generally the poorest. There are shading artifacts across the interior of the model.

The speed (isnt as fast as the environment map, faster than the model augmentation, well have to see how it compares to the two pass method.)

This method needs pixel shader hardware and the associated extensions.

Again, only trivial additional memory is needed for textures.

## 9 Model Augmentation

During the preprocessing stage quads are generated along each edge formed by pairs of triangles. Information on normals for both generating triangles and an averaged direction (from the normals) are embedded in properties of the quads and a new model is made of just the quads.

During runtime at each vertex a vector to the eye is dotted with both the normals. All quads start degenerate. If the signs of the dot products are different (the quad is on the silhouette) the vertex is offset in the averaged direction by a preset amount. Bases of quads are always extended by 0. Quads not on the silhouette stay degenerate and dont get drawn. The original triangle model is used to occlude any concavities on the back side that may have caused quads to extend.

Time to write and debug the code was medium. 28 lines of vertex shader assembly, half a dozen lines to load the shader and prep constants, 2 lines of code in the main display loop and an estimated two dozen lines for the quad model generation that was based on the normal model reader.

The visual quality of this method is generally very good. However because it uses quads along the vertex, there can be cracks in the silhouette where the quads dont meet on the outer edge.

This method depends on vertex shader hardware and the associated extensions.

In addition to the triangle model additional space is needed for the degenerate quad model. (1.5X the size of triangle model for the quad model)

---

## 10 Future Work: Resolution Independent Silhouette Rendering

For polygonal models the silhouette set for a given viewpoint can be computed using one of the methods described in Subsection 4 and placed into a line list. These lines could then be projected to screen space on a point by point basis and drawn at the necessary resolution in post script. Curves can be treated in a similar manner but could be made to appear smooth by sampling at each desired resolution. Gooch presented a method for multiresolution silhouette curves for NURBS surfaces[11]. Finkelstein and Salesin presented a method for drawing multiresolution curved strokes using Bezier curves[9]. This method can be applied to both polygonal and continuous surfaces.

The unique problem presented by multiresolution silhouette rendering is occlusion culling. There are currently three common methods for performing occlusion culling on the silhouette set. The first, an image space method, is to simply render the model over the previously rendered silhouettes. This method while applicable, does not seem appropriate for the purpose of multiresolution silhouettes. The second method for occlusion culling of the silhouette set, presented by Northrop et al.[24], is to render the silhouettes, using the image space occlusion culling method, with uniquely coded line colors. Then by checking these color ID's edges in object space, only the edges which need to be drawn can be determined. The problem with this method is that it is extremely slow. Models with fewer than 200 silhouette edges take 5-10 seconds to process. The third method for performing occlusion culling on the silhouette set is Appel's algorithm which is used by Markosian et al.[23]. The drawbacks to using Appel's algorithm are code complexity and slow speed.

## References

- [1] BENICHO, F., AND ELBER, G. Output sensitive extraction of silhouettes from polygonal geometry. In *Pacific Graphics '99* (Seoul, Korea, October 1999).
- [2] BOTTINO, A., AND LAURENTINI, A. Experimenting with nonintrusive motion capture in a virtual environment. *The Visual Computer* 17, 1 (2001), 14–29. ISSN 0178-2789.
- [3] BUCHANAN, J. W., AND SOUSA, M. C. The edge buffer: A data structure for easy silhouette rendering. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering* (June 2000), ACM SIGGRAPH / Eurographics, pp. 39–42.
- [4] CHUNG, Y. C., PARK, J. W., SHIN, H., AND CHOI, B. K. Modeling the surface swept by a generalized cutter for nc verification. *Computer-aided Design* 30, 8 (1998), 587–594.
- [5] CURTIS, C. Loose and sketchy animation. SIGGRAPH 1998 Session Notes.
- [6] D.J., B., AND J.F., H. Rapid approximate silhouette rendering of implicit surfaces. In *Proceedings of Implicit Surfaces 98* (June 1998), pp. 155–164.
- [7] ELBER, G., AND COHEN, E. Hidden curve removal for free form surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (Dallas, Texas, August 1990), vol. 24, pp. 95–104. ISBN 0-201-50933-4.
- [8] EVERITT, C. One-pass silhouette rendering with geforce and geforce2. NVIDIA Corporation White Paper.
- [9] FINKELSTEIN, A., AND SALESIN, D. H. Multiresolution curves. In *Proceedings of SIGGRAPH 94* (Orlando, Florida, July 1994), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / ACM Press, pp. 261–268. ISBN 0-89791-667-0.
- [10] FUA, P., PLANKERS, R., AND THALMANN, D. From synthesis to analysis: Fitting human animation models to image data. In *Computer Graphics International '99* (June 1999), IEEE CS Press. ISBN ISBN 0-7695-018.
- [11] GOOCH, A. A. Interactive non-photorealistic technical illustration. Master's thesis, University of Utah, December 1998.

- 
- [12] GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P. S., AND RIESENFELD, R. Interactive technical illustration. In *1999 ACM Symposium on Interactive 3D Graphics* (April 1999), ACM SIGGRAPH, pp. 31–38. ISBN 1-58113-082-1.
- [13] HAINES, E. Soft planar shadows using plateaus. *Journal of Graphics Tools* 6, 1 (2001), 19–27.
- [14] HERTZMANN, A., AND ZORIN, D. Illustrating smooth surfaces. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Computer Graphics Proceedings, Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 517–526. ISBN 1-58113-208-5.
- [15] JASON MITCHELL. Real-time non-photorealistic rendering. ATI Corporation White Paper.
- [16] JASON MITCHELL. Real-time image-space outlining for non-photorealistic rendering - siggraph 2002 session notes. ATI Corporation White Paper.
- [17] JENSEN, C. G., RED, W. E., AND PI, J. Tool selection for five-axis curvature matched machining. *Computer-Aided Design* 34, 3 (March 2002), 251–266. ISSN 0010-4485.
- [18] JOHNSON, D. E., AND COHEN, E. Spatialized normal cone hierarchies. In *2001 ACM Symposium on Interactive 3D Graphics* (March 2001), pp. 129–134. ISBN 1-58113-292-1.
- [19] KETTNER, L., AND WELZL, E. Contour edge analysis for polyhedron projections. In *Geometric Modeling: Theory and Practice* (1997), Springer, pp. 379–394.
- [20] LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering* (June 2000), ACM SIGGRAPH / Eurographics, pp. 13–20.
- [21] LEE, W., GU, J., AND MAGNENAT-THALMANN, N. Generating animatable 3d virtual humans from photographs. *Computer Graphics Forum* 19, 3 (August 2000). ISSN 1067-7055.
- [22] LUM, E., AND MA, K.-L. Hardware-accelerated parallel non-photorealistic volume rendering. In *NPAR 2002* (June 2002), ACM SIGGRAPH / Eurographics.
- [23] MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 97* (Los Angeles, California, August 1997), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 415–420. ISBN 0-89791-896-7.
- [24] NORTHRUP, J. D., AND MARKOSIAN, L. Artistic silhouettes: A hybrid approach. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering* (June 2000), ACM SIGGRAPH / Eurographics, pp. 31–38.
- [25] POP, M., DUNCAN, C., BAREQUET, G., GOODRICH, M., HUANG, W., AND KUMAR, S. Efficient perspective-accurate silhouette computation and applications. In *Proceedings of the seventeenth annual symposium on Computational geometry* (2001), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 60 – 68. ISBN:1-58113-357-X.
- [26] RASKAR, R. Hardware support for non-photorealistic rendering. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware (HWWS)* (August 2001), Proc ACM Press, pp. 410–46i.3. ISBN 1-58113-407-X.
- [27] RASKAR, R., AND COHEN, M. F. Image precision silhouette edges. In *1999 ACM Symposium on Interactive 3D Graphics* (April 1999), ACM SIGGRAPH, pp. 135–140. ISBN 1-58113-082-1.

- 
- [28] RHEINGANS, P., AND EBERT, D. Volume illustration: nonphotorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics* 7, 3 (July - September 2001), 253–264. ISSN 1077-2626.
- [29] SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. Silhouette clipping. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Computer Graphics Proceedings, Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 327–334. ISBN 1-58113-208-5.