Silhouette Rendering Based On Stability Measurement

John Brosz* Faramarz Samavati† Mario Costa Sousa‡

The University of Calgary

Abstract

A better silhouette for a mesh can be rendered if we take into account the stability of edges inside and outside the current silhouette. Using the dot product between the normal and the viewing direction we can measure this stability. This gives us two types of edges: silhouette and non-silhouette and an associated stability of each. We apply this classification and stability measure to achieve several different styles of rendering as well as temporal frame coherence.

Keywords: Silhouette Stability, Quality Silhouette Extraction, Non-photorealistic Rendering, Temporal Coherence.

1 Introduction

The silhouettes lines on an object that we are rendering are very important. In traditional art these lines improve the perception of the mass of an object [Crane 1900][Whitaker 1994]. Frequently artists present us with simplified representations of objects using just the silhouette lines as in sketches, figures, and ink drawings.

The usefulness of silhouettes in helping us see three dimension shapes on a two dimensional medium make silhouette lines very important in computer graphics. The use of silhouettes by artists further makes these lines critical when performing non-photorealistic rendering (NPR). When extracting and displaying silhouettes there are still several open problems in dealing with stability, temporal coherence, artifacts, and stylization.

*e-mail: brosz@cpsc.ucalgary.ca

†e-mail:samavati@cpsc.ucalgary.ca

‡e-mail:mario@cpsc.ucalgary.ca

In this work we look exclusively at object space extraction of silhouette from polygon meshes. The reason for this concentration is that object space silhouettes give us a one pass solution, as opposed to image space silhouettes where the two rendering passes occur. Two rendering passes are necessary as the first rendering is used to extract the silhouette and the second rendering is used to display the processing [Saito and Takahashi 1990]. We are only concerning ourselves with polygonal meshes because they are currently the most used objects for rendering and because their silhouettes most often suffer from stability and coherence problems.

Our work in this paper finds a stability measure for every edge of a polygonal mesh. Using these stability measures we find additional edges that can be considered part of the silhouette. The resulting silhouette graph highlights areas of the mesh where the silhouette may be present. This can be helpful in: reducing the temporal artifacts when animating, adaptively subdividing silhouette areas to find better silhouettes, performing precise ink drawings, or applying ink textures to the mesh. In introducing these stability measures we make the following contributions:

- We give an explanation of silhouette edge stability and its relevance to extracting silhouette from meshes.
- We show how to extract silhouettes in such a way that any data structure containing edge information can be used to find our silhouette stability measure.
- We provide several different methods of applying stylization to edges in order to achieve NPR renderings.
- We obtain silhouettes that are frame coherent.

Section 2 describes work in the areas of silhouette extraction and silhouette frame coherence. Section 3 overviews our algorithm for extracting silhouettes and finding edge stability measures. In Section 4 we provide the basis for our stability measure. Section 5 provides several methods of rendering based on stability measures and Section 6 discusses the results achieved by these renderings. Lastly Section 7 provides our conclusions and ideas for future work.

2 Related Work

Our work in this paper is related to two main types of silhouette operations. The first is efficiently extracting high quality silhouettes. That is, extracting the best possible silhouette from a mesh. The second silhouette related operation is achieving temporal coherence when renderings of the silhouette are animated.

Current work with object space extraction of silhouettes from polygonal meshes has yielded many good results but these methods have problems with artifacts and great difficulties in achieving temporal coherence when animating. Methods that attempt to handle errors in the silhouettes use inefficient case checks to check all points in the silhouette to find errors and deal with them. In other methods the silhouette is created without using actual edges of the polygon meshes. These methods have troubles performing visibility culling.

Quality Extraction of Silhouettes
Work has been done on quality extraction of silhouettes by Pop et al[2001]. In this work silhouettes are extracted through use of a pre-calculated dual space. Their dual transform relates a point (x,y,z) with a plane dual $ax+by+cz+1=0$ and a plane $ax+by+cz+d=0$ with the point dual $(a/d,b/d,c/d)$[Pop et al. 2001]. An edge in this dual space is a line segment connecting the two points that are the duals of the planes tangent to the faces touching the edge. Silhouette edges are found by searching for edge duals that intersect the dual of the viewpoint. In order to perform this operation quickly an Oct-tree or BAR tree is precomputed and used.

The work done by Pop et al has two major strengths: it is fast and efficient method of finding every silhouette edge in the scene and it is very good at tracking changes to the silhouette as the viewpoint changes. This algorithm also offers support for tracking non-silhouette edges that are close to becoming silhouette edges. In our work we have chosen to use the edge buffer[Buchanan and Sousa 2000] to find silhouette rather than the faster dual plane because we must test every edge for every frame of animation which would be costly with an Oct or Barr tree. Pop et al's approach does not address any methods to remedy artifacts present in the extracted silhouette.

The second work on quality extraction of silhouettes that we are reviewing is that done by Hertzmann and Zorin [2000]. A slightly different dual space is used to extract silhouette edges. The major contribution of the work by Hertzmann and Zorin that we wish to discuss is their method of increasing the quality of the extracted silhouette. Rather than simply detecting the edges that separate a front-facing face and a back-facing face, new edges are created. All edges where the dot product of the vertex normal and the viewing direction changes from positive to negative between the two vertices are marked as having a silhouette pass through them. The point where this dot product is equal to zero is linearly interpolated between the two vertices' dot products. The resulting points are connected to one another forming piecewise linear silhouette line.

The silhouettes constructed out of this method are better silhouette edges than those that follow the polygonal mesh's edges. Our approach does not do any such interpolation, instead we use more existing edges (making visibility culling easier) to indicate areas where a silhouette may be found. This approach gives a solution that is more general and stable and assists in applying several different stylizations.

Kirsanov et al[2003] describe an algorithm that creates a silhouette for a high-resolution mesh that is similar in structure to the silhouette of a coarse version of the same mesh. The method is based on loop decomposition and the resulting silhouette is made of the actual edges of the original model without the redundancy present on the silhouette of the high resolution mesh. Their approach cannot guarantee the temporal coherence of the silhouette approximation.

Another method of achieving better silhouettes is through processing the extracted silhouette and filtering out the silhouette artifacts. Northrup and Markosian[2000] do this by projecting the silhouette edges to image space and removing overlapping parts of the silhouettes and parts of the silhouette that are considered undesirable based on fixed rules. Isenberg et al[2002] remove errors from chains of silhouette edges by projecting into the Z-Buffer and using this depth information to look for several error cases. These methods both achieve some measure of better looking silhouettes but frame coherence achieved by these results can be worse than standard results due to cases of error filtering suddenly being applied to subsequent frames.

Frame Coherence
An exciting property of our work is that it obtains a degree of frame coherence. The most impressive work on frame coherence of silhouettes is that done by Kalnins et al[2003]. In this work the extracted silhouette edges at the key frames are chained into silhouette loops and converted into parameterized brush paths that are then rendered as stroke primitives. Brush paths in the initial frame of animation are decided by silhouette paths and in subsequent frames brush paths are indirectly determined by the closest paths from previous frames. Brush paths are most often parameterized by a scaled arc length and using this parameterization stroke styles can be applied for NPR effects. Between key frames the silhouettes are adjusted by a linear interpolation between the equivalent

sample points on the two parameterizations of the silhouette.

Our work considers the unaddressed issue of why silhouette lines change and where these lines come from. By addressing these topics a robust solution to temporal coherence is achieved. Our system also boasts simpler visibility testing without use of an offset buffer.

Masuch et al[1998] and Bourdev[1998] have also done work on achieving temporal coherence in line drawings and non-photorealistic strokes respectively. These two works are concerned with methods of maintaining the same parameterization and ensuring the same stroke is applied to the same line in subsequent frames of animation. In our work we are able to avoid such problems because our stylizations are more simple. We are merely drawing the edges of polygons rather than applying parametric stylizations.

3 Overview

For each image rendered the following steps are taken:

- A mesh is loaded into our data structure. The exact data structure is not important so long as it stores each edge in the mesh. For our implementation we have used the edge buffer [Buchanan and Sousa 2000].
- We use the viewing direction to test the faces and store the state of each edge. For each edge we store the state of the faces bordering the edge (i.e., front-facing or back-facing), a bit indicating whether the edge's stability is greater than the relevant stability threshold(T_n or T_s), and a float containing the edge's stability measure.
- The gathered information is then used to apply the chosen stylization method and the mesh is rendered.

4 Silhouette Edges

When dealing with polygonal meshes, we generally define the silhouette lines as paths following silhouette edges. A silhouette edge is an edge shared by a front-facing polygon and a back-facing polygon. These silhouette edges include not only the outline of the object (the contour), but also internal silhouettes within the object. We classify the orientation of the mesh's polygons by examining the dot product of the polygon normal, N and the viewing direction, V (a vector from the camera to the viewing plane). When both of these vectors are normalized, we end up with a result ranging from -1.0 to +1.0. If $N \cdot V = 0$ the polygon perpendicular to the viewing direction and the entire face is on the silhouette. Otherwise if $N \cdot V > 0$ the face is front-facing and if $N \cdot V < 0$ the face is back-facing.

The general object space algorithm for detecting silhouette edges is to classify all the faces as front or back facing, determine which edges share front and back faces, and then perform the desired silhouette effects or exaggeration to these edges. Our chosen efficient data structure for doing this is the edge buffer [Buchanan and Sousa 2000]. It should be noted that simply drawing such silhouette lines will result in obscured silhouettes (i.e., silhouettes on the opposite side of the object) being drawn. This is handled by hidden line removal (also know as visibility culling) and is discussed later on in this paper.

4.1 Silhouette Errors

Polygonal meshes are discrete approximations of the model's actual surface(s). The result of this fact that the silhouette edges we detect are only approximations of the real silhouette of the model. The set of silhouette edges we extract are limited to polygonal edges when compared to the model's actual silhouette because our silhouette only occurs on the edges of the model's polygons. This results in three major types of artifacts present in the mesh (as described by [Foster et al. 2004]): jumping errors, zig-zag errors, and off-track errors. Jumping errors occurs when the silhouette moves between two lines of polygons as in figure 1.

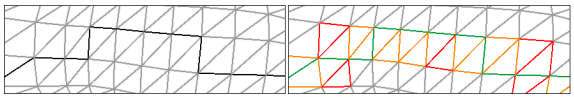

Figure 1: An example of a jumping error on left and our solution on right. Color legend is shown in Figure 2.

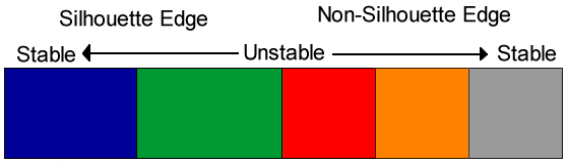

Figure 2: Coloring legend.

Zig-zag errors occur when more than one edge of a single polygon is classified as a silhouette edge as in figure 3. Off-track edges are edges that technically are silhouette edges according to the value of the dot product described above, but are artistically poor. These are often caused by the coarseness of the mesh and the numerical instability

of the operations performed. Figure 4 shows an off-track error.

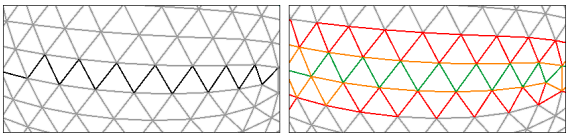

Figure 3: An example of a zig-zag error and our solution on right. Coloring is same as before.

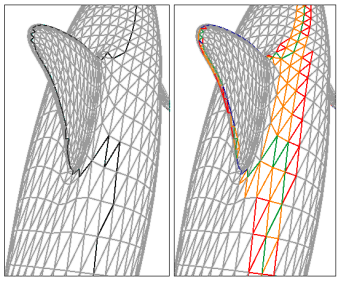

Figure 4: An example of a off-track error and our solution on right. Coloring is same as before.

These artifacts become more pronounced when the mesh is animated. As the viewing direction and the face normals change, edges may appear and disappear abruptly between frames of animation. Most of these artifacts result from edges popping in and out of the silhouette. These popping effects are distracting and confusing to viewers. Such effects are demonstrated in figures 5 and 16.

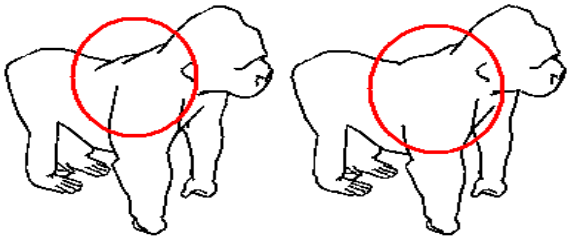

Figure 5: A small rotation of the ape model results in the circled edges suddenly disappearing from the image.

4.2 Stability of Silhouette Edges

Our proposed solution to the artifacts present in current silhouette extraction techniques is to measure each silhouette edge's instability. Using these stability measures we find additional edges that can be considered part of the silhouette. The resulting silhouette graph highlights areas of the mesh where the silhouette may be present. In Figures 1 ,3 and 4, this area is represented by different colors whose interpretation is provided in 2. The area together with the stability measure help to avoid silhouette artifacts.

First of all, however, we must decide when a silhouette edge is deemed to be unstable. We can say that a silhouette edge is stable if this edge is unlikely to disappear from the silhouette if a small change is made to the viewing direction, orientation or position of the mesh's vertices. Conversely, unstable silhouette edges are those edge that are more likely to disappear with these same changes. We must also consider non-silhouette edges. Unstable non-silhouette edges are those that are likely to pop into the silhouette, and stable non-silhouette edges are those not likely to pop into the silhouette. Once we have an idea of how stable a particular silhouette edge is we can render the edge in a fashion where stable edges form the bulk of the image and unstable edges merely fill in faint details.

4.2.1 Polygon Stability

To obtain this estimate of stability for an edge we look at the polygons sharing that edge. If a polygon is likely to change state (from being front-facing to being back-facing or vice versa) we declare the edges adjacent to the polygon to be unstable. To determine whether the polygon is likely to change state, we examine the dot product between the viewing vector and the polygon normal. Consider a polygon where the dot product value is close to zero. It is clear to see that slight changes to the view point, the model, or the model's orientation, can easily change the polygon's orientation from front-facing to back-facing or vice versa. This makes this polygon an unstable polygon. In the case where the dot product result is further away from zero, we find that this polygon is unlikely to change its orientation.

4.2.2 Edge Stability

Now we consider the edges of the mesh. An edge shared by two polygons that are both stable are considered as either stable silhouette edges or stable non-silhouette edges, depending on the orientation of the two polygons. Consider the case where an edge is shared by a front-facing polygon and a back-facing polygon where one or both of the polygons is unstable. In this case the edge is currently a silhouette edge but could easily become a non-silhouette edge. We consider this edge an unstable silhouette edge. Lastly consider the case where an edge is shared by two forward-facing polygons or two backward-facing polygons and at least one polygons has an unstable dot product result. This tells us that a slight shift in view-

ing direction or polygon orientation could make this edge a silhouette edge. Therefore we consider this to be an unstable non-silhouette edge. Table 1 presents the possible edge classifications.

Table 1: Stability Classifications Of Edges

Edge Type	Dot Product	Orientation Of Adjacent Polygons
Stable Sil.	$\|N \cdot V\| > T_s$ for both polys.	F,B
Unstable Sil.	$\|N \cdot V\| < T_s$ for at least one poly.	F,B
Unstable Non-Sil.	$\|N \cdot V\| < T_n$ for at least one poly.	F,F or B,B
Stable Non-Sil.	$\|N \cdot V\| > T_n$ for both polys.	F,F or B,B

F = Front facing polygon, B = Back facing polygon.

4.2.3 Controlling Stability

Next we must consider what threshold (T) value marks the boundary where polygons should be considered unstable. We must also note that there are two separate thresholds we should consider. The first is T_s, a threshold for silhouette edges, and the second we will refer to as T_n, a threshold for non-silhouette edges.

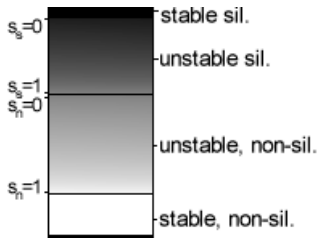

Figure 6: Portrayal of our stability based silhouette coloring scheme. The range of the unstable silhouette and non-silhouette colored areas depend on the thresholds T_s and T_n. Note that $s_s = \frac{|N \cdot V|}{T_s}$ and $s_n = \frac{|N \cdot V|}{T_n}$.

High values (> 0.20) when used for T_s result in all silhouette edges becoming marked as unstable. This becomes more and more the case when we are dealing with fine meshes where the surface is quite smooth. Lower values of T_s were found to give a good indication of the edges that were likely to pop out of the image. We found values in the interval $[0.05, 0.10]$ gave good results for all meshes. Unless otherwise stated in our figures we have used $T_s = 0.07$. Figure 7 shows the useful indicator of instability that $T_s = 0.07$ yields. When examining T_n we have found that the ideal value for this depends greatly upon how the user desires to use the unstable non-silouette edges and marginally on the model you are using. In the local ink stroke rendering described later in the paper, we found T_n = [0.05-0.70] useful for creating ink outlines of images. We found $T_n = [0.9 - 1.0]$ useful for filled ink drawings. Examples of different T values being applied are shown in figure 8.

4.3 Computational Cost

One benefit of this instability measure is that it is computationally very inexpensive. All of the expensive calculations (e.g., finding the polygon normals, normalization, etc) are already calculated for finding extracting silhouettes. Finding the stability of the polygons requires storing each polygon's dot product value's difference from zero and comparing this value to T, requiring a very small slow-down. Tracking the stability of each edge requires comparing the stability values for the two polygons. The cost of these comparisons is very small. The cost of storing these stability values when done in a naive manner is $O(e) + O(p)$ (where e = number of edges and p = number of polygons in the mesh).

5 Rendering

From observing ink drawings we have noticed that sharp and well defined edges tend to be marked sharply. Furthermore, we found that areas of the silhouette that are gently curved tend to be colored lighter and have ink applied over a larger area. Our method can been seen as marking sections of the mesh where an artist might draw a silhouette. The areas of the silhouette that are unstable are areas where the artist is more likely draw more darkly and the unstable areas are places where the artist is likely to draw less darkly. We can mimic how far the artist wishes to extend the silhouette by changing T_n.

We also note that our approach of considering near silhouette edges is much like that of DeCarlo et al [2003] in using suggestive contours. As is mentioned in this work, artists often exaggerate contours, extending them beyond the technically existing contour where $N \cdot V = 0$ to a contour that would be seen from a different viewpoint.

After determining measures of instability for the silhouette edges, we need to decide upon how we wish to render these stable and unstable silhouette edges. In our silhouette extraction application, we have experimented with several different rendering methods to exploit the usefulness of this stability measurement.

5.1 Locally Rendering Unstable Edges

The first and least interesting rendering method is to simply render the stable, unstable, and non-silhouette lines

Figure 7: The ape model rotated with $T_s = 0.07$ and $T_n = 0.00$. Stable silhouette edges are shown in black, unstable are shown in gray. Notice that the edges turn gray before they disappear from the silhouette.

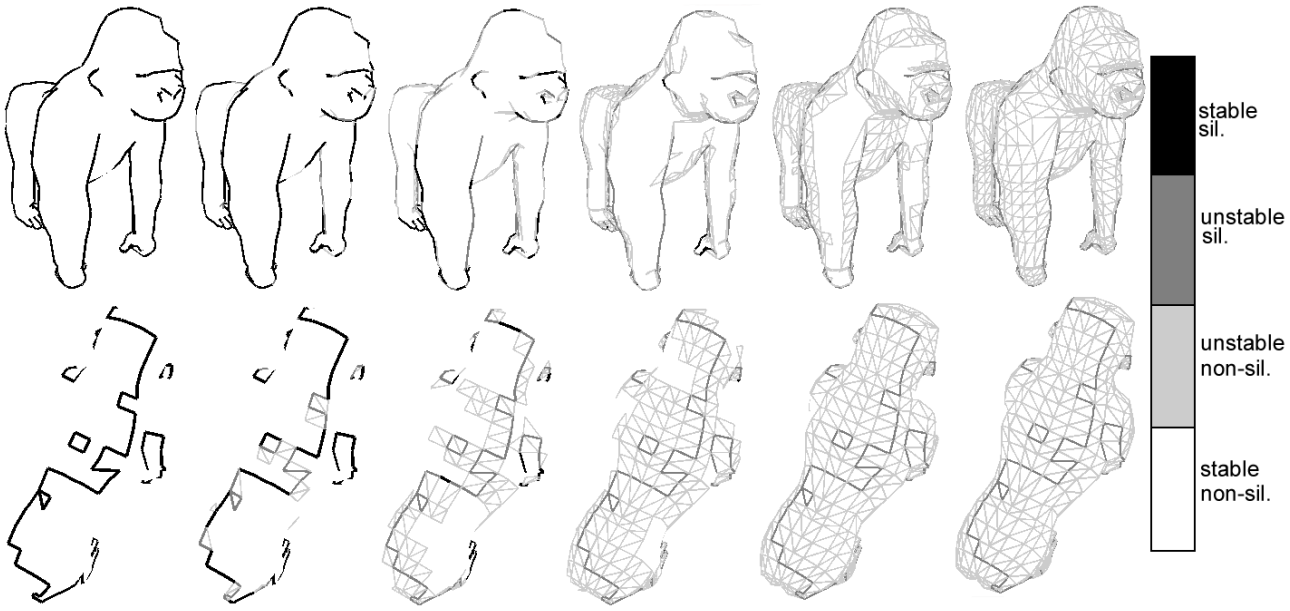

Figure 8: A model of an ape with show with $T_s = T_n = 0.0, 0.5, 0.25, 0.50, 0.75, 1.0$ from left to right. Bottom images are of the silhouette fixed and the model rotated. The traditionally extracted silhouette starts as shaded blue (stable) and becomes green (unstable) as T_s becomes larger until the entire original silhouette is considered unstable. This is generally undesirable and is the reason why we usually limit T_s to a value in the range $[0.05 - 0.10]$.

in black. This rendering style only yields useful results while $T_n < 0.20$ by adding a few edges to extend the silhouette lines.

A more interesting local rendering method is obtained by drawing stable edges in black with a wide(r) width and the unstable edges as shades of grey with smaller widths. The formulations of the widths and colors are shown in table 2. This method yields a more aesthetically pleasing results. Even better results can be achieved with finer meshes or when the mesh has been subdivided as in figure 9.

Table 2: Local Method Coloring Scheme

Edge Type	Color	Width
Stable Sil.	0	max
Unstable Sil.	$\frac{1}{2}(1-s)$	$\frac{1}{2}max(1+s)$
Unstable Non-Sil.	$\frac{1}{2}(1+s)$	$\frac{1}{2}max(1-s)$

$s = (\frac{|N_i \cdot V|}{T})$ s.t. N_i is the normal of the least stable polygon touching the edge and T is either T_n or T_s based on edge type. We are referring to color as scale from 0 (black) to 1 (white).

5.2 Applying Shading To Faces

An alternative to drawing lines, it is also possible to calculate the average shade of each vertex and then draw the polygon faces according to our previous shading scheme. The result of this shading when $T_n = 1.0$ is exactly the

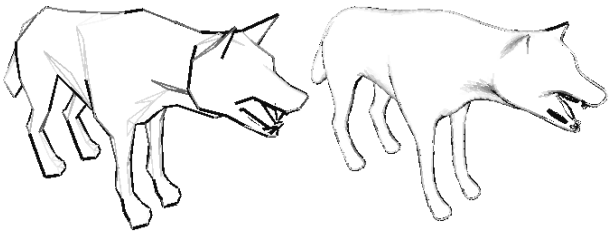

Figure 9: Locally shaded silhouettes of a wolf ($T_n = 0.40$). The left image is the silhouette of the coarse mesh (550 $\triangle s$), the right image is the silhouette of the model after two levels of subdivision (9K $\triangle s$).

same as applying perfect diffuse Phong lighting on a light grey object with a light source fixed at the eye position (see figure 13). This makes sense as the diffuse shading used in the Phong lighting equations is based entirely on $|N \cdot L|$ and when $L = V$ our shading methods and the Phong lighting method produce the same results.

Results that are more unique are obtained with smaller values of T_n. Only faces where all three edges have been marked as being silhouette or unstable are drawn. In these situations, we get slight shading effects around the edges of the silhouette as seen in figure 13.

5.3 Texture Mapping Ink Strokes

The next use of our stability measures is texture mapping pen and ink tones to the faces of the mesh. We have taken the ink tones shown in figure 10 and texture mapped pieces of those tones to faces of the mesh.

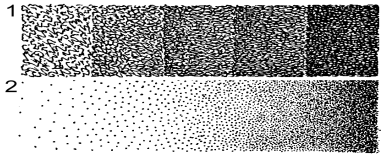

Figure 10: The two collections of ink textures from used in the ink stroke texture mapping.

We choose darker tones for the faces that are more stable and lighter tones for faces that are less stable. As before, only faces where all three edges are part of the silhouette or unstable are drawn. The stability of each face is chosen to be the same as the stability of the most stable edge of the face, this creates a "darker" drawing than would otherwise be achieved with use of the average or minimum stability of the face's edges. The texture coordinates are generated using a simple linear mapping. We experimented with two different ink tones and typical results of such renderings are shown in figures 11 and 12.

Figure 11: Model of a gorilla with the first set of texture tones (10K $\triangle s, T_n = 0.40$).

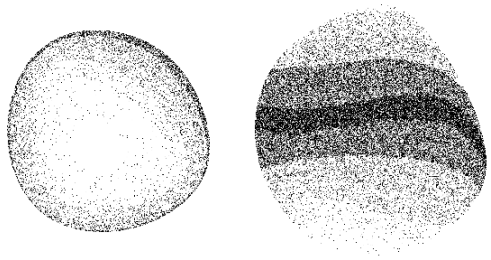

Figure 12: A silhouette textured spheroid on the left. On the right the silhouette has been locked in place and then the model has been rotated to better illustrate the grades texture shading approaching the silhouette (T_n = 1.00).

5.4 Filled Ink Stroke Rendering

Another use we have found for our silhouette stability measures is to use these measures to create filled ink strokes as described by Sousa et al [2003]. In this paper a filled ink stroke is described as ribbon a, b, b', a' where (a, b) are the vertices of a polygonal mesh and (a', b') is found "by extruding its vertices (a, b) in the direction of their normals (n_1, n_2)[Sousa et al. 2003]. The amount of extrusion in originally used in [Sousa et al. 2003] is based upon the curvature of the mesh multiplied by a user defined amount. One can think of the area of the silhouette edges as the area of the mesh an artist might place a silhouette stroke in. For our purposes we have based the amount of extrusion upon the stability of the edge multiplied by a user defined amount (max). The exact amount of displacement used for each type of silhouette edge is outlined in table 3. All edges are drawn in black. Example results are shown in figure 15.

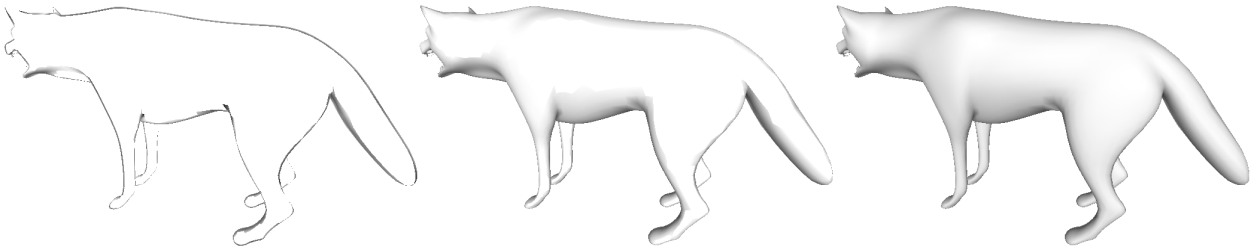

Figure 13: The silhouette-shaded faces of a subdivided wolf model ($9K \triangle s, T_n = 0.3, 0.8, 1.0$ from left to right).

Table 3: Precise ink displacement amounts.

Silhouette Edge Type	Displacement
Stable	max
Unstable	$(0.7 + 0.3 * s) * max$
Unstable Non-Sil.	$0.1 + 0.6 * (1 - s) * max$

$s = \frac{|N_i \cdot V|}{T}$ s.t. N_i is the normal of the least stable polygon touching the edge.

5.5 Subdivision

One problem with our local rendering method is that coarse meshes suffer from artifacts when T_n is larger than 0.05 (depending on the mesh). These artifacts are long edges that stretch further into the center of the object than is reasonable or reveal the faces of the mesh. For an example see the left image of figure 9. A reasonable way to counter this is through use of subdivision to create smaller edges and faces. Using Loop subdivision[Schroeder and Zorin 1998] we subdivide the faces of the mesh that contain unstable edges making the faces smaller so they are less noticeable in the final image. This is shown in right image of figure 9.

5.6 Hidden Line Removal

An important task when rendering the silhouettes we have detected is in ensuring that the obscured silhouette lines on the back of the mesh are not shown to the user. In our system we achieve this by rendering the mesh in the object's chosen fill color with the vertices of its polygons displaced a small fraction along the reverse of the vertex normal. This is the same approach as used by Sousa [2003]. An example of silhouettes with and without hidden line removal is shown in figure 14

5.7 Temporal Coherence

An important side benefit of rendering based on the stability of each silhouette edge is that we achieve a temporal coherence when the mesh is animated or rotated. This coherence is due to the fact that silhouette edges no longer suddenly appear or disappear from the silhouette. Instead they are gradually shown as they become more stable and fade from view as they become less stable. A single silhouette line is, due to their unstable nature of the dot product over a surface, very unstable. Instead of finessing the line to interpolate between different states as in [Kalnins et al. 2003] we represent the surrounding edges of the mesh that will replace the silhouette line as in 16. This gives us a more stable solution.

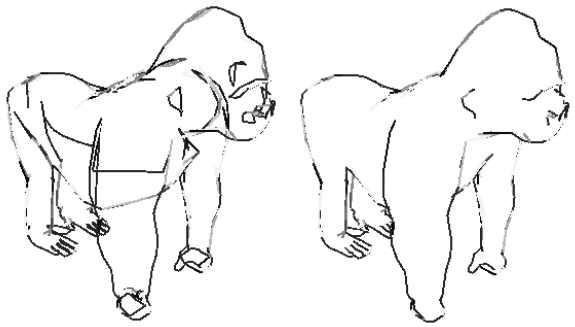

Figure 14: A model of a gorilla with (right) and without (left) hidden line removal.

6 Results and Discussion

We selected eight meshes ranging from 500 to 268,000 triangles. Our system achieved fast rendering rates with real time performance for models of less than 30,000 triangles and at interactive rates with models of less than 200,000 triangles. These results were obtained on a Athlon XP2200 with a GeForce2 GTS graphics card.

Local Rendering: For fine meshes our local rendering technique provides results that are more pleasing than traditional results. Figure 9 shows such results. These local rendering results are more pleasing than traditional methods since some simple shading is performed.

Texture Mapping Ink Strokes: The result achieved by the texture mapping achieves an ink drawing like rendering. The first texture collection was found to be most

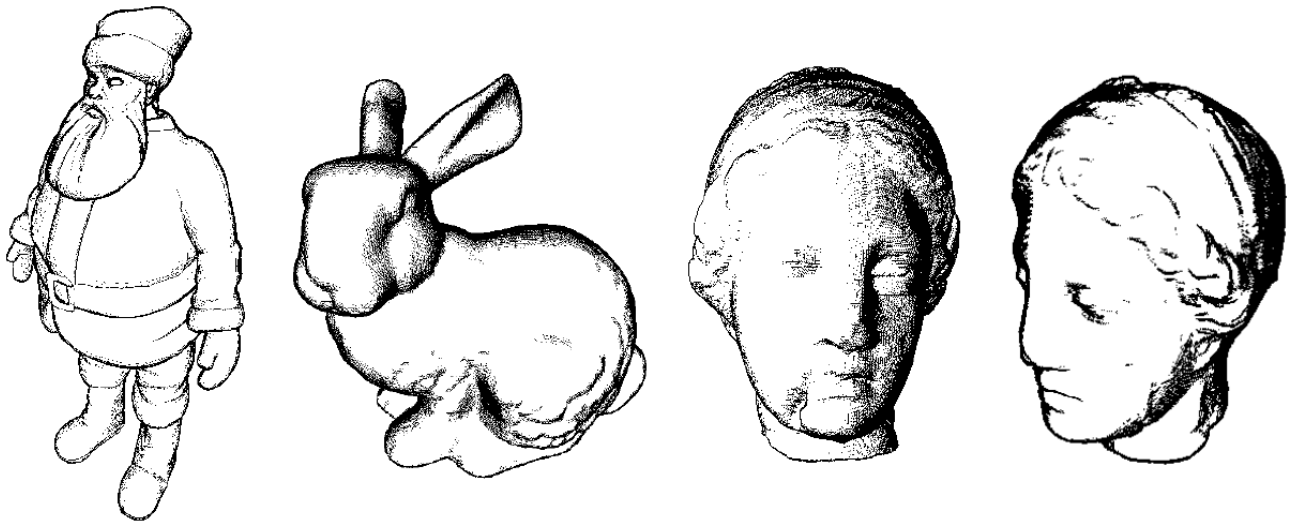

Figure 15: Ink stroke rendering with different silhouette stability measure T_n. From left to right: Santa Claus ($51K \triangle s, Tn = 0.64$), the Stanford bunny ($70K \triangle s, T_n = 0.70$) and the Igea artifact ($268K \triangle s, T_n = 0.85, 0.60$).

useful with T_n values in the $0.40 - 0.60$ range, the second collection gave the best results for $T_n = 0.9 - 1.0$. Renderings are real-time for polygonal meshes of less than 10,000 faces and can be viewed at interactive speeds for meshes of larger sizes.

Filled Ink Stroke Rendering: The visual result achieved by the precise ink rendering style using our stability measures is comparable to that of the more time consuming measures used in [Sousa et al. 2003]. This ink stroke method requires additional time compared to our local rendering method due to the calculation of vertex normals. The rendering time also increases as polygons are now being rendered (instead of lines).

Temporal Coherence: We have found that by coloring and texturing our models based on edge stability we achieve temporal coherence as seen in figure 16.

7 Conclusions And Future Work

We have developed a 3D silhouette extraction system that provides us with higher quality silhouettes. We do this by measuring the stability of edges based upon the stability of dot product between the viewing direction and face normals. This stability measure provides a please method of shading, a guide for apply NPR ink textures and strokes, and can be used to achieve temporal coherence. These silhouettes include more edges and do not suffer from distracting artifacts when animated. These stability measures are very inexpensive provided exhaustive silhouette detection has already been performed. Additionally we have applied a two different methods of NPR rendering based upon our stability measure. We have achieved fast and good looking results with temporal coherence.

In our future work we would like to consider a better measure of relative stability within the classes of edges. Our current measure of relative stability (referred to as s in Tables 2 and 3) considers an edge between two flat faces that are close to parallel with the viewing angle as stable as an edge with one face close to parallel with the viewing angle and the other edge more perpendicular to the viewing angle. We would like to find a measure that separates these instabilities. We also believe that varying the threshold (T_n) across the mesh based upon local mesh measures could yield interesting results.

In our research we have also experimented with adaptive subdivision. We use the adaptive subdivision to limit subdivision to the faces containing unstable edges. It may be possible that over many subdivisions of a given mesh the cost of many levels of adaptive subdivision, resulting in fewer faces, will be less than testing every single face of the mesh when dealing with extremely large meshes. We have not found scenarios where this is the case, but future work should be done in this area.

We feel that our stability measure is an inexpensive and useful technique that can be used with many more existing NPR techniques. Our future work will include applying our stability measures to more NPR techniques.

Acknowledgements
Our special thanks go to Kevin Foster for providing us with the original edge buffer silhouette extraction code that our system is based upon. We are also very grateful to Ruth Hart-Budd for her editing contributions.

Figure 16: Model of Beethoven's bust being rotated. Top row shows the silhouette extracted without considering stability, the bottom row shows our rendering with $T_n = 0.40$.

References

BOURDEV, L. 1998. *Rendering Nonphotorealistic Strokes with Temporal and Arc-Length Coherence*. Master's thesis, Brown University.

BUCHANAN, J., AND SOUSA, M. C. 2000. The edge buffer: a data structure for easy silhouette rendering. In *Proceedings of the first international symposium on non-photorealistic animation and rendering*, 39–42.

CRANE, W. 1900. *Line Form*. George Bell Sons, London.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics (TOG) 22*, 3, 848–855.

FOSTER, K., SOUSA, M. C., AND SAMAVATI, F. 2004. Multiresolution for polygonal silhouette error correction. Accepted to ICCSA 2004.

HERTZMANN, A., AND ZERIN, D. 2000. Illustrating smooth surfaces. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 517–526.

ISENBERG, T., HALPER, N., AND STROTHOTTE, T. 2002. Stylizing silhouettes at interactive rate: From silhouette edges to silhouette strokes. *Computer Graphics Forum 21*, 3, 249–258.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics (TOG) 22*, 3, 856–861.

KIRSANOV, D., SANDER, P. V., AND GORTLER, S. J. 2003. Simple silhouettes over complex surfaces. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on geometry processing*, 102–106.

MASUCH, M., SCHUMANN, L., AND SCHLECHTWEG, S. 1998. Frame-to-frame coherent line drawings for illustrative purposes. In *Proceedings of Simulation und Visualisierung '98*, SCS Europe, 101–112.

NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic silhouettes: A hybrid approach. In *Proceedings of NPAR '00*, 31–37.

POP, M., DUNCAN, C., BAREQUET, G., GOODRICH, M., HUANG, W., AND KUMAR, S. 2001. Efficient perspective-accurate silhouette computation and applications. In *Proceedings of the seventheenth annual symposium on Computation Geometry*, 60–68.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. In *Proc. of SIGGRAPH '90*, 197–206.

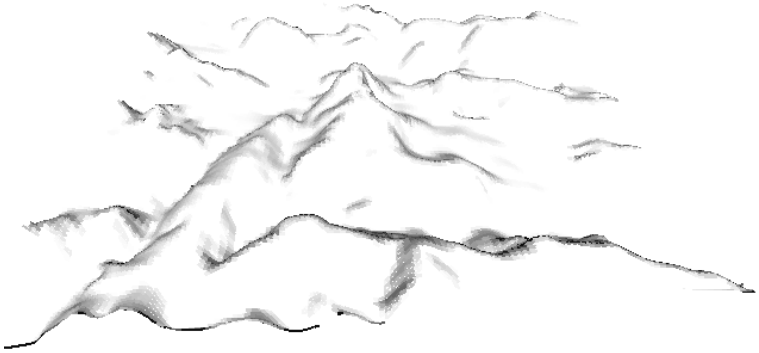

Figure 17: Terrain rendered with local shading ($49K$ $\triangle s, T_n = 0.35$).

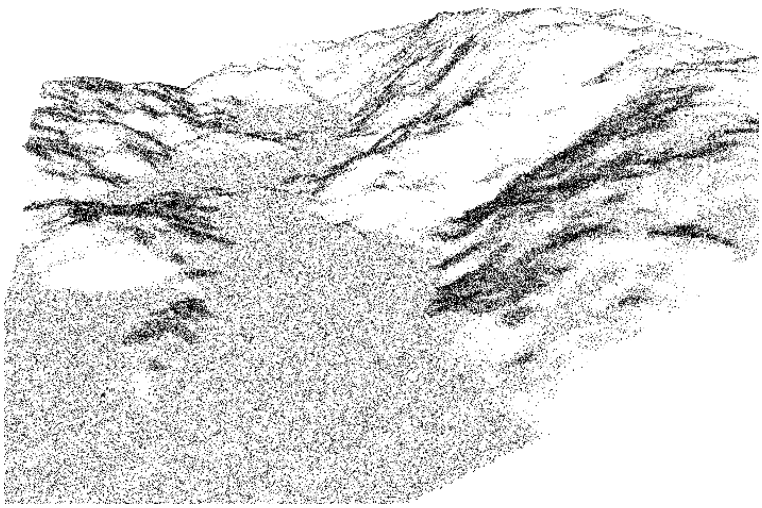

Figure 18: Terrain rendered with ink texture mapping ($33K$ $\triangle s, T_n = 0.90$).

SCHROEDER, P., AND ZORIN, D. 1998. Subdivision for modeling and animation. In *ACM SIGGRAPH Course Notes*, vol. 12.

SOUSA, M. C., FOSTER, K., WYVIL, B., AND SAMAVATI, F. 2003. Precise ink drawing of 3d models. *Computer Graphics Forum 22*, 3, 369–379.

WHITAKER, S. 1994. *The Encyclopedia of Cartooning Techniques*. Running Press, Philadelphia.