Eurographics Symposium on Geometry Processing(2003)
L. Kobbelt, P. Schröder, H. Hoppe (Editors)

Simple Silhouettes for Complex Surfaces

D. Kirsanov, P. V. Sander, and S. J. Gortler

Harvard University

Abstract

Complex meshes tend to have intricate, detailed silhouettes. This paper proposes two algorithms for extracting a simpler, approximate silhouette from a high-resolution model. Our methods preserve the important features of the silhouette by using the silhouette of a coarser, simplified mesh as a guide. Our simple silhouettes have significantly fewer edges than the original silhouette, while still preserving its appearance.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometric algorithms.

1. Introduction

Silhouettes play an important role in several areas of Computer Graphics, such as interactive visualization, non-photorealistic rendering and shadow computation. There are several systems that draw silhouettes in various ways to help depict geometric models [Gooch et al. 98, Gooch et al. 99, Hertzmann et al. 00, Markosian et al. 97, and many others]. Unfortunately, many high-resolution geometric meshes often have overly complicated silhouettes. For example, see Figure 5(a2,b2), which shows the silhouette of the bunny model as viewed from both the eye position as well as an auxiliary side view; this silhouette contains a large number of loops, many intertwined loops, and loops that are quite wiggly. This silhouette complexity is usually unnecessary and sometimes costly, both in terms of speed and quality of rendering algorithms applied on silhouette edges.

In this paper, we propose methods for computing "approximate silhouettes" that attempt to represent the most salient features of the silhouette, while suppressing the extra detail and complexity. Our basic approach is to first compute the actual silhouette of a smoothed coarse geometric mesh that approximates the original model (see Figure 5(col. 1)). This "coarse silhouette" typically has a simpler structure. Our goal is to create a silhouette for the fine mesh that is similar in structure to the coarse silhouette. This results in a high resolution, but "simple" silhouette (see Figure 5(cols. 3,4)).

We have explored a few approaches, and in this paper we describe two different methods that present various tradeoffs.

1.1 Previous work

There have been a number of interesting approaches to efficiently compute silhouettes on models. Sander et al. [2000] build a spatial hierarchy that allows one to quickly dismiss subsets of mesh edges as being non-silhouette. Barequet et al. [1999] and Hertzmann and Zorin [2000] describe methods that also build a hierarchy, but do their computation in dual space. Barequet et al. also describe how their method can be used incrementally to more quickly recompute silhouettes under small viewpoint changes. Markosian et al. [1997] propose a method to find silhouettes by tracking them from frame to frame as well as randomly testing for the creation of new silhouette loops. This method tends to find the larger loops more reliably.

Northrup et al. [2000] address the simplification of the silhouettes in the image space by linking the visible segments into the chains, thus resulting in a simpler, approximate silhouette. Hertzmann and Zorin [2000] describe an elegant way of finding simple approximate silhouettes on meshes. Silhouettes found by this method are quite smooth and are always comprised as a set of non-intersecting loops. In their method, the silhouette edges all span across mesh triangles and are not original mesh edges.

In our methods the silhouette is composed of actual mesh edges. Furthermore, redundant short loops are usually removed from our computed simple silhouettes.

2. Approach

The input to our system is a high-resolution fine mesh. From this mesh, we create a progressive mesh [Hoppe et al. 1996]. We then extract a simpler, coarser mesh, and apply smoothing on it in order to prevent artificially generated silhouettes.

During runtime, for a given viewpoint, the objective is to create a silhouette for the fine mesh that is similar in structure to that of the coarse mesh. In this section we will describe two approaches to achieve this objective. In particular we will describe one method in which the set of the edges in the coarse silhouette are constrained to be some subset of the actual silhouette edges. In this method, loops from the actual silhouette are iteratively included into the simple silhouette as long as they sufficiently match the coarse silhouette.

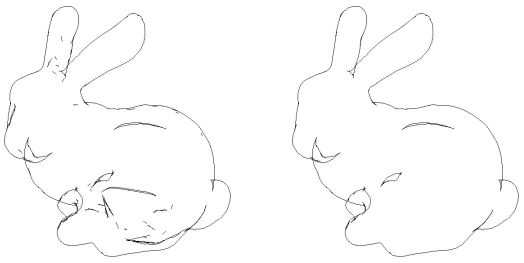

Figure 1: *Fine silhouette (left) and simple silhouette (right) computed using the loop picking algorithm. The simple silhouette consists of only four loops.*

The second method constrains the simple silhouette to be structurally similar to the coarse silhouette. For each coarse-silhouette loop, a corresponding loop is found on the original mesh. With this second approach, it may be necessary to sometimes include non-silhouette edges in the simple silhouette.

Next we will describe how we decompose the silhouette into loops – a sub-step that is required for both methods, – and then we present the two silhouette extraction methods.

2.1 Loop Decomposition

On a smooth surface, the silhouette is comprised of a set of disjoint closed loops. On a triangle mesh, these loops can intersect (for example, a silhouette vertex can be incident to four silhouette edges) creating a slightly more complicated structure. Our first step will be to take a complicated silhouette and describe it as the union of (possibly intersecting) loops.

Given a viewpoint, a silhouette edge is an edge that is adjacent to one front-facing triangle and one back-facing triangle. Silhouette edges are directed edges, where the left adjacent face is front-facing. A silhouette vertex is a vertex adjacent to a silhouette edge. In particular, on a triangle mesh, the faces around a silhouette vertex can be partitioned into an alternating set of front-facing and back-facing triangles, thus the number of silhouette edges adjacent to a silhouette vertex must be *even*.

Since silhouette vertices have even "silhouette edge valence," we can decompose the silhouette into loops. We seek a decomposition that yields loops without self-intersections.

We achieve this by starting with an arbitrary vertex and walking along the directed silhouette edges until we get back to the original vertex, thus closing the loop. We repeat this process until all silhouette edges have been assigned to loops. In order to accelerate the walking algorithm, we use a hash table of directed silhouette edges indexed by the source vertices [Sander et al. 00].

During this walk, we may get to intersections where we have pick between two or more edges to follow. When that happens, we just arbitrarily pick any one of the possible edges. If at any point during a walk we encounter a sub-loop, we "detach" it and store it as a separate loop.

This method does not produce a unique loop decomposition, as it is sensitive to the location where the walk starts and the decisions made at silhouette intersections. However, this greedy approach is extremely fast, and by snapping off sub-loops, it achieves our objective of constructing loops without self-intersections.

2.2 Method A: Loop picking

In this approach, we first extract the silhouette of both the coarse and fine meshes using the algorithm from Sander et al. [2000]. We then perform loop decomposition on the fine silhouette, yielding a set of silhouette loops.

Our goal is to pick a subset of the fine silhouette loops that minimizes an error metric based on distance to the coarse mesh silhouette. More specifically, the metric is the sum of the squared distances between each coarse silhouette vertex, and its closest point on the fine silhouette. The loops are picked one by one in greedy fashion, minimizing this metric. We terminate when the error gets below a specified threshold that is proportional to the amount of detail that is desired.

To update the error, after a loop is added, every coarse silhouette vertex must be checked to see if it is closer to one of the newly added fine silhouette vertices. In order to do this efficiently, we first pre-compute the distances from every coarse silhouette vertex to its closest vertex on each of the fine silhouette loops, and store these distances in a 2D array. When evaluating a candidate loop, we just do look-ups to this array to compute the error.

Applying the above algorithm results in a set of long silhouette loops that resemble the silhouette of the coarse mesh. Short loops and loops that are not geometrically close to a coarse mesh silhouette edge are not picked because they do not significantly decrease the error. Very short loops are pruned and discarded a priori in order to make the search more efficient.

Figures 1 and 5(col. 3) show examples of simple silhouettes extracted using the above algorithm. For performance results, refer to the Section 3.

2.3 Method B: Loop mapping

In the previous method, we started with the fine silhouette, and selected loops based on their proximity to the coarse silhouette. In this method, we instead start with the coarse silhouette, and try to create loops over the fine mesh that resemble the coarse loops. For every loop of the coarse silhouette we find a corresponding loop on the fine mesh. The goal of this method is to retain all major features of the silhouette and ensure continuity when the model is rotated. By meeting this goal we do not guarantee that all edges of the fine loop belong to the fine silhouette.

The first step of this algorithm is to decompose the coarse silhouette onto non-self-intersecting loops using the method described in Section 2.1. We also need a mapping between coarse and fine models, i.e. for each fine triangle we need to know the corresponding coarse triangle (Figure 2). To create such a mapping, we simplify the mesh using half-edge collapses. Each coarse mesh edge corresponds to the shortest

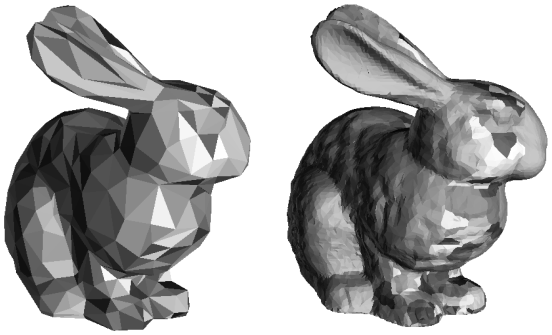

Figure 2: *Coarse and fine bands are shown with colored triangles on the bunny modes. In order to show the mapping between models, each triangle on the coarse mesh (left) has the same color as corresponding fine triangles on the fine mesh (right).*

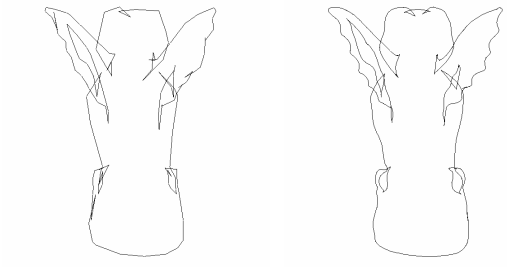

Figure 3: *One loop of the coarse silhouette of the gargoyle (left) and its mapping on the fine mesh (right).*

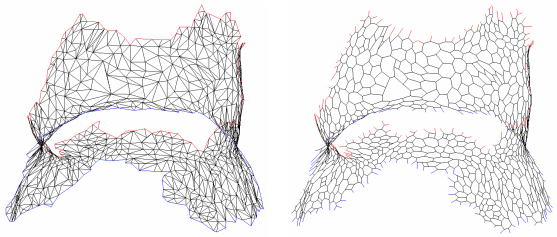

Figure 4: *Primal (left) and dual (right) graphs. Internal boundary (red) on the primal graph corresponds to the source edges on the dual graph. External boundary (blue) on the primal graph corresponds to the sink edges on the dual graph.*

path of fine mesh edges between the two analogous vertices on the fine mesh. Therefore, each face can then be trivially mapped to a set of faces on the fine mesh.

We define the *coarse band* of a coarse loop as all triangles that have at least one common vertex with this loop. The coarse loop splits the coarse band into two parts, internal and external. The outer boundary of the internal part of the band is called *internal boundary*, the outer boundary of the external part of the band is called *external boundary*. The internal and external boundaries of the coarse band could consist of more than one loop, but this does not affect our algorithm.

We can map the coarse band from the coarse mesh onto the fine band on the fine mesh (Figure 2). Note that internal and external boundaries of the coarse band are mapped to the internal and external boundaries of the fine band.

Our goal is to approximate the coarse loop with the best possible loop in the fine band. To do this, we find a minimal-cost loop of edges over the fine mesh that separates the internal and external boundaries. In order to measure the cost of an edge, the simplest approach would be to set its weight to be equal to its length. In this case, we will be looking for the shortest loop possible. To ensure that the fine loop is close to the fine silhouette, we multiply the weights of the fine edges that are on the fine silhouette by the small parameter α. Increasing the value of this parameter makes the loop smoother and decreasing forces it to include more edges from the actual silhouette. In our experiments we set $\alpha = 0.1$. Figure 3 shows a loop from the coarse silhouette and its corresponding loop on the fine mesh.

Now we have to find a minimal path on the weighted graph that partitions it into two parts. This is a standard problem that can be easily solved with the minimal cut approach. First, we construct the dual graph (Figure 4). That is, we associate with each face of the band f_i a dual vertex F_i. We also create a single source vertex R and a sink vertex K. With each non-boundary edge e_i bounding two faces f_j and f_k, we associate a dual edge E_i that connects F_j and F_k. For each edge e_i belonging to the external boundary we associate a dual edge E_i which connects the sink K with the single face that e_i bounds. For each edge e_i belonging to the internal boundary we associate a dual edge E_i which connects the source R with the single face that e_i bounds. We set the capacity of each dual edge the weight of the associated primal edge $c(E_i)= w_i$.

We define a cut of the dual graph as a partition of the vertices F_i into two sets F and Q with $R \in F$ and $K \in Q$. The cost of a cut is the sum of the capacities of the edges between these sets. It can be shown that the edges across the minimal cut in the dual graph correspond to dual to the edges of the minimal separating path in the primal graph [Buehler et al. 2002]. Informally the reason why this theorem is true is because a cut of the dual graph is a partition of the dual vertices into to two sets. This cut then corresponds to a partition of the primal faces into two spatial regions. The dual edges across the cut correspond to the primal edges forming the boundary between these two spatial regions. Thus there is a natural duality between cuts and paths.

In summary, the entire algorithm can be described as follows:

Preprocess:

- Simplify original mesh and establish mapping between the simplified coarse and the original fine mesh.

Runtime:

- Decompose coarse silhouette into loops.
- For every coarse loop:
- Calculate the coarse band.
- Find the internal and external boundaries of the band.
- Find the corresponding fine band and its boundaries.

- Compute the weights of the edges and create a dual graph.
- Compute the min-cut of the dual graph.
- Find the corresponding fine loop.

3. Results

We implemented both algorithms, and applied them to several models. Table 1 shows the mesh resolutions and timings of our algorithm on the bunny and gargoyle models. Figure 5 shows silhouette renderings for both methods. We also show side views of the silhouette, in order to demonstrate how intricate the original fine mesh silhouette is.

For silhouette extraction, we used an optimized version of the algorithm from Sander et al. [2000]. It extracts the silhouette of a 20,000-face bunny mesh in half a millisecond. The loop decomposition step from Section 2.1 also takes approximately half a millisecond for that model.

The total silhouette extraction time for the loop picking algorithm is 5 and 10 milliseconds for the bunny and gargoyle models, respectively. So, we can compute the simple silhouette of a 20,000-face bunny model at a rate of 200 frames/sec. Note that this does not include model and silhouette rendering time.

The loop mapping algorithm is significantly slower, running at a rate of 2 to 5 frames/sec. However, it results in a rendering with far fewer silhouette edges, while still resembling the fine mesh silhouette, as shown in Figure 5.

	bunny	gargoyle
# coarse mesh faces # fine mesh faces	1,000 20,000	1,000 30,000
coarse mesh sil. extraction time (ms) + loop decomposition	0.038 0.098	0.042 0.102
fine mesh sil. extraction time (ms) + loop decomposition	0.553 1.051	0.917 1.602
Loop picking total time (ms)	5.172	9.582
Loop mapping total time (ms)	190.000	521.000

Table 1: *Quantitative results (Pentium 4, 2.0Mhz).*

4. Summary

In this paper, we described two algorithms to extract a simple silhouette from a high-resolution mesh by using the silhouette of a coarse, simplified mesh as a guide. The silhouette produced by both of these methods are composed by actual edges of the original model. The loop picking algorithm is very efficient and extracts a subset of the actual silhouette edges. The loop mapping algorithm is slow, but it is able to capture all the important features of the silhouette with far fewer loops, thus further eliminating the redundancy present on the silhouette of high resolution meshes. Another advantage of this method is that only small amount of the edges of the fine mesh has to be processed. The major limitation of both proposed approaches is that we cannot guarantee the temporal coherence of the silhouette approximation for the moving models.

References

BAREQUET, G., DUNCAN, C., GOODRICH, T., KUMAR, S., AND POP, M. Efficient Perspective-Accurate Silhouette Computation. *Symposium on Computational Geometry 1999*, 60 – 68.

BUEHLER, C., GORTLER, S. J., COHEN, AND M., MCMMILLAN, L. Minimal Surfaces for Stereo Vision. *ECCV 2002*, 885 – 899.

GOOCH, A., GOOCH B., SHIRLEY P., AND COHEN, E. A non-photorealistic lighting model for automatic technical illustration. *SIGGRAPH 1998*.

GOOCH, B., SLOAN, P., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. Interactive technical illustration. *ACM Symposium on Interactive 3D graphics 1999*, 31 – 38.

HERTZMANN, A., AND ZORIN, D. Illustrating smooth surfaces. *SIGGRAPH 2000*, 517 – 526.

HOPPE, H. Progressive meshes. *SIGGRAPH 1996*, 99 – 108.

NORTHRUP, J.D., MARKOSIAN, L., Artistic Silhouettes: A Hybrid Approach. *NPAR 2000*, 31 – 38.

MARKOSIAN, L., KOWALSKI, M., TRYCHIN, S., AND HUGUES, J. R Real time non photorealistic rendering. *SIGGRAPH 1997*, 415 – 420.

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. Silhouette clipping. *SIGGRAPH 2000*, 327 – 334.

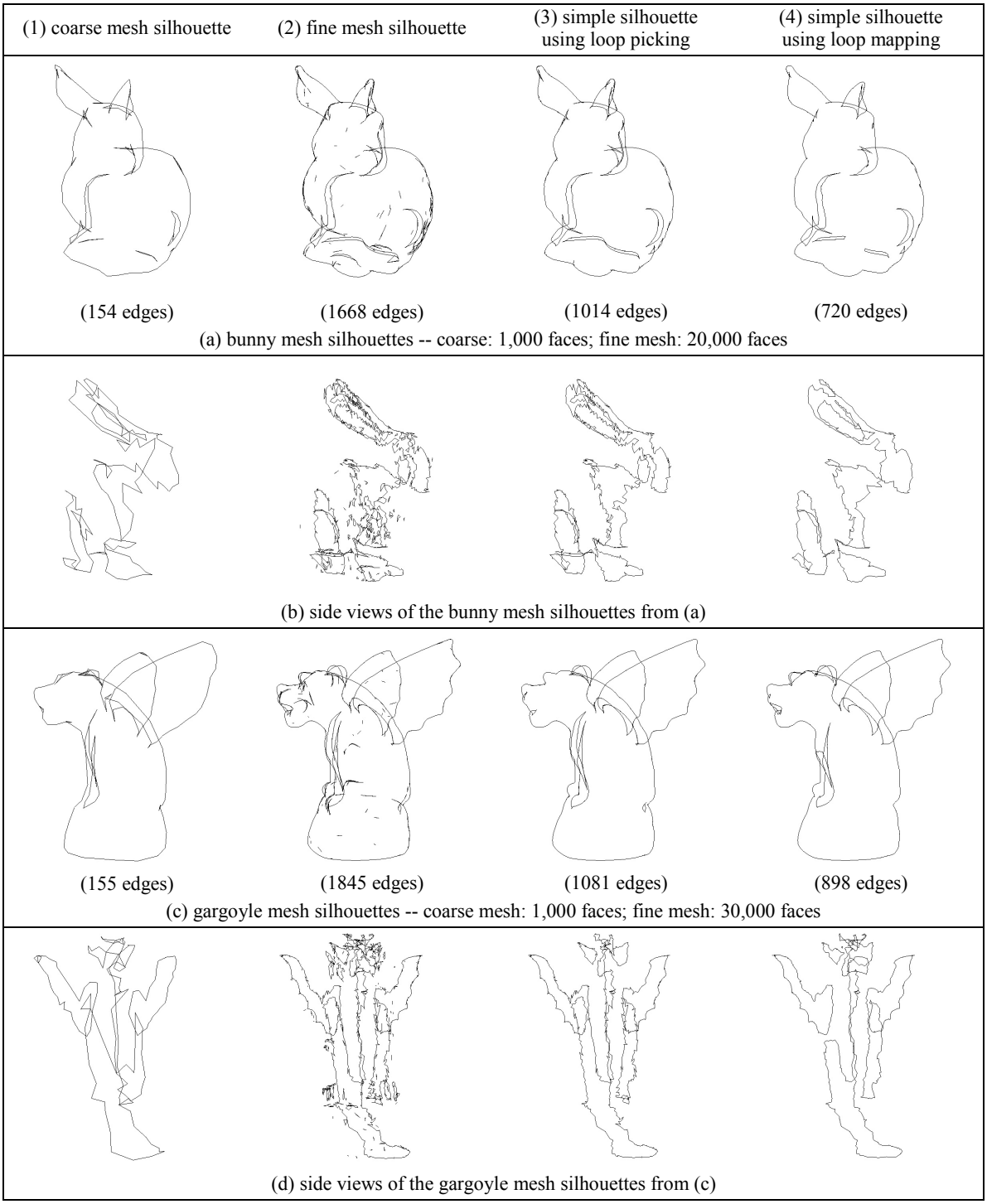

Figure 5: *Silhouette rendering comparisons of the bunny and gargoyle meshes.*