Efficient Perspective-Accurate Silhouette Computation and Applications*

Mihai Pop
9712 Medical Center Dr.
Rockville, MD 20850
mpop@tigr.org

Gill Barequet
Faculty of Computer Science
The Technion—IIT
Haifa 32000, Israel
barequet@cs.technion.ac.il

Christian A. Duncan
Dept. of Computer Science
University of Miami
Coral Gables, FL 33124
duncan@cs.miami.edu

Michael T. Goodrich
Wenjing Huang
Subodh Kumar
Center for Algorithm Engineering
Dept. of Computer Science
Johns Hopkins University, Baltimore, MD 21218
[goodrich|hwenjing|subodh]@cs.jhu.edu

ABSTRACT

Silhouettes are perceptually and geometrically salient features of geometric models. Hence a number of graphics and visualization applications need to find them to aid further processing. The efficient computation of silhouettes, especially in the context of perspective projection, is known to be difficult. This paper presents a novel efficient and practical algorithm to compute silhouettes from a sequence of viewpoints under perspective projection. Parallel projection is a special case of this algorithm. Our approach is based on a point-plane duality in three dimensions, which allows an efficient computation of the *changes* in the silhouette of a polygonal model between consecutive frames. In addition, we present several applications of our technique to problems from computer graphics and medical visualization. We also provide experimental data that show the efficiency of our approach.

Keywords

Rendering, silhouette, simplification.

*Work of all authors on this paper has been supported by ARO MURI Grant DAAH04-96-1-0013. Work by the second author was supported also by a grant from the Smoler Research Fund and by a Fialkow Academic Lectureship. Work by the fourth author was supported also by NSF Grants CCR-9732300 and PHY-9980044. Work by the sixth author was supported also by NSF Career Award CCR-9733827 and by NSF grant ERC-9731748.

SCG'01, June 3-5, 2001, Medford, Massachusetts, USA.

1. INTRODUCTION

Efficient computation of silhouettes of polyhedra is a central problem in many applications, such as model simplification [15, 16, 19, 22, 30], image-based rendering [25, 29], collision detection [4], shadow computation [8], nonphotorealistic rendering [5, 12, 14, 23], and computer animation [13]. Since silhouette edges represent discontinuities in the visibility of an object, they are useful features in the registration of models in two and three dimensions, a technique used in medical robotics to align preoperative models to X-ray images [21]. Perceptual importance of silhouettes also allows to visualize large polygonal models based on their silhouettes [29].

We assume throughout this paper that the surface polygons of a solid model are specified consistently so that all facet-normals point outwards (or all point inwards) from the model. We also assume that the model is a 2-manifold. While nonmanifold edges may be better classified based on additional geometric rules, we conservatively mark them as silhouette edges in our current implementation. For a 2-manifold without boundaries, our algorithm computes precisely the edges that are on the silhouette.

Every edge of the silhouette of a polyhedral model is shared by two facets with opposite orientations with respect to the viewpoint. That is, an edge is a silhouette edge if one of the adjacent facet-normals points toward the viewpoint and the other normal points away from it. In Figure 1 the edge e is on the silhouette of the model since the facet f_1 is facing away from the viewer while facet f_2 is facing toward the viewer. Figure 2 shows a polyhedral representation of a chess bishop, where the silhouette edges are highlighted with thick lines.

Silhouette computation has recently received special attention [2, 3, 11, 14, 17, 28, 29]. Raskar and Cohen [28] treat the problem from a computer graphics point of view. They use the rendering pipeline to draw silhouette edges on the screen. Their algorithm draws enlarged backfacing polygons first, followed by (normal-sized) frontfacing polygons to

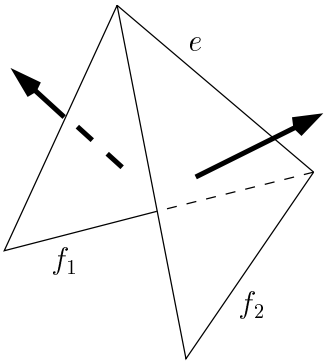


Figure 1: A silhouette edge

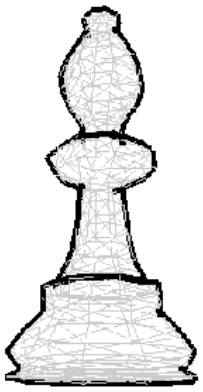


Figure 2: A silhouette of a chess bishop

produce the silhouette. While the problem of hidden-edge removal adds considerable complexity to silhouette-finding algorithms, their approach has the advantage of showing only visible silhouette edges. However, the rendering-based approach does not explicitly compute silhouette edges; it rather just renders them through image processing techniques. In many applications we want to obtain a list of silhouette edges (for example, for model registration).

Benichou and Elber [3] solve this problem under parallel projection. In this case the visibility of a facet from the viewpoint (located at infinity) is uniquely defined by the facet's normal. The authors project all the facet normals onto the unit sphere, and map the edges of the polyhedron into arcs on the sphere. Finding the silhouette edges is then equivalent to finding which spherical arcs are intersected by a plane that passes through the center of the sphere. Benichou and Elber solve this problem by further projecting the sphere onto a unit cube, thus reducing it to finding line-segment intersections in the plane. The latter problem is solved by using known techniques [1, 7, 24]. The main drawback of this approach is that it only works for parallel projections, whereas most practical applications require the use of perspective projections (e.g., complex scene rendering and image registration). Our algorithm addresses the shortcomings of both approaches, by explicitly computing silhouette edges under perspective projection.

Two competing algorithms were recently introduced in [29] and in [14]. In the first work silhouettes are computed by locating an anchored cone of normals in an n-ary tree. In the second work the silhouette of a polygonal model is computed by intersecting the dual of the model with the dual plane of the viewpoint. The plane-surface intersection needed in this work is, in general, a harder and more unstable problem than the point-location operations needed in our work. In addition, our algorithm is based on tracking only the changes of the silhouette rather than computing all the silhouette edges in each frame. Also, the tree construction, maintenance, and search are simpler in our method. As a result, our method is more efficient both in theory and in practice.

The simplest brute-force approach for computing silhouettes is to check the facets adjacent to every edge. Such an approach must examine the whole model. However, it has the advantage that the computation done for each edge is quite fast since it involves only a couple of dot-products operations. Using a more complicated procedure is beneficial only if the number of silhouette edges is considerably smaller than the number of edges in the model. Kettner and Welzl [17] examine the complexity of the silhouettes of polyhedral models. They show that the number of silhouette edges of an ε-approximation of the unit sphere under the Hausdorff distance is proportional to $1/\sqrt{\varepsilon}$, whereas the total number of edges of the approximation is on the order of $1/\varepsilon$. Thus we expect to have roughly $O(\sqrt{n})$ silhouette edges for approximations of smooth convex objects with n edges. For nonconvex objects the number of silhouette edges can be higher, mainly due to the complexity of a convex partition of the object. Kettner and Welzl show that the fraction of silhouette edges for some commonly-used models varies between 2% and 40%, where the higher values correspond to more complex objects. For most objects the value is around 10%, which indicates that a careful algorithm can yield an improvement over the brute-force approach. We also note that only a few silhouette edges change when the viewpoint moves slightly. The brute-force algorithm cannot take advantage of this fact and recomputes all the silhouette edges for each frame.

In contrast, our approach focuses on the edges that either become silhouette edges or cease being silhouette edges in each frame. Thus we update a small number of "silhouette flags" at each frame. We reduce the problem of updating the silhouette to finding edges that cross a (three-dimensional) double-wedge of two planes. The advantages of this scheme are manifold:

- The silhouette computation is efficient;
- The silhouettes are object-space precise and are perspective accurate;
- It is able to compute edges that are 'close' to being on the silhouette; and
- It is able to compute silhouettes from a set of viewpoints (e.g., along a line segment or within a triangle).

2. THE ALGORITHM

We present a novel approach for finding silhouettes of polyhedral models. Our technique makes use of a duality between points and planes in three dimensions.

The dual transform associates a point $P = (a, b, c)$ with the plane dual(P) given by $ax + by + cz + 1 = 0$. Conversely, a plane π with equation $ax + by + cz + d = 0$ is associated

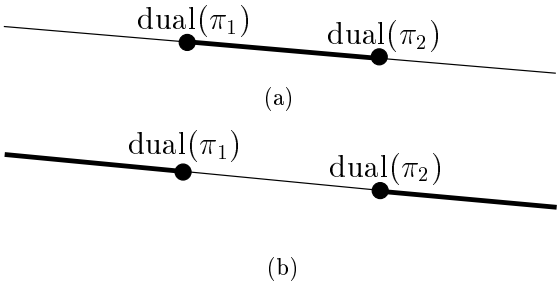


Figure 3: Two possible duals of an edge shared by two facets supported by the planes π_1 and π_2 (shown with thick lines)

with the point dual(π) = $(a/d, b/d, c/d)$. Note that if P is the origin ($a = b = c = 0$) (resp., π passes through the origin ($d = 0$)), then the dual plane (resp., the dual point) is at infinity.

It is well known that a point v lies in a plane π if and only if the point dual(π) lies in the plane dual(v). We use this simple fact for finding silhouette edges. Consider the dual of an edge e of the model. Let f_1 and f_2 be the facets sharing the edge e, and let π_1 and π_2 be the supporting planes of these two facets. We define dual(e) as the shortest path in the dual space taken by dual(π_1) as π_1 is rotated about e toward π_2 on the shortest path (smaller rotation angle in the primal space). Saying differently, the normal to π_1 is rotated about e towards the normal of π_2 along the shortest arc. Note that the two normals, and all the intermediate positions thereof, lie in a plane perpendicular to e.

The following theorem, which is a rather known fact, proves that the dual of this motion occurs on a line connecting the points dual(π_1) and dual(π_2). Note that π_1 may pass through the origin while it is rotated toward π_2. In this case dual(e) passes through a point at infinity. This dual edge is the *difference* between the entire line defined by dual(π_1) and dual(π_2), and the segment connecting the two points. The two situations are depicted in Figure 3.

It is also well known [27] that:

THEOREM 1. *Given a line ℓ in the three dimensional space, the duals of all planes that contain ℓ lie on a line.*

This follows directly from the fact that the duality preserves occurrences. We will now prove that dual edges uniquely define silhouette edges, thus establishing the correctness of our algorithm.

THEOREM 2. *An edge e is on the silhouette as seen from a viewpoint v, if and only if* dual(e) *intersects the plane* dual(v).

PROOF. The claim follows from the definition of the dual of the edge e. If e is a silhouette edge, then its adjacent facets f_1 and f_2 have different orientations with respect to the viewpoint. Therefore, while rotating one facet (or rather, rotating its supporting plane) toward the other about e, we must cross the viewpoint, hence the dual segment intersects with the dual viewplane.

Conversely, if a dual segment intersects with the dual viewplane, then the primal facets corresponding to the endpoints of the edge must have the viewpoint on different sides (since by moving from one to the other we need to cross the viewpoint); therefore the primal edge is, by definition, a silhouette edge. □

2.1 Silhouette Update

In order to find the silhouette edges of a model, it is enough to construct the duals of the edges of the model, then report those edges whose duals intersect the dual of the viewpoint. The problem of finding the set of three-dimensional segments intersected by a query plane is difficult, although its two-dimensional variant has been studied [1, 7, 24] extensively. Even if we were able to efficiently compute the answer to such queries, we would need to recompute the complete silhouette every time the viewpoint changes. An on-line silhouette-finding algorithm needs to compute the change in the silhouette between frames. We note that as the viewpoint moves, the collection of silhouette edges changes whenever the dual of the viewpoint intersects one of the points dual to the supporting planes of the facets of the model. Maintaining the collection of silhouette edges while the viewpoint is in motion thus leads to the following problem, which is a generalization of the segment-dragging problem [6, 10, 18]:

> Given a set of points in three dimensions and a plane π, report the points hit by π while it moves continuously (performing rotations and translations).

The segment-dragging problem has no simple efficient solution known even in two dimensions when the segment is not axis parallel. The two-dimensional solution relies on constructing a simplicial partition with a low *stabbing number* for the set of points. (The stabbing number of a line segment is the number of other segments or regions it intersects.) The segment-dragging problem is then answered by carefully traversing this partition. The efficiency of the algorithm is guaranteed by the bound on the stabbing number of the partition. However, this procedure does not extend easily to three dimensions. Therefore, the implementation of an approach based on a simplicial partition is impractical. At the same time there are a few practical limitations. Since the change of the position of the viewpoint is discrete (at least in our application), our algorithm receives the positions of the viewpoint at instants determined by the frame rate of the graphical interface. We could apply our procedure by interpolating the position of the viewpoint between the key positions. However, this turns out to be too slow as the algorithm must handle each of the possibly many points crossed by the moving viewplane. Hence, this theoretically-efficient solution is not quite practical.

As a remedy for these problems we make use of a heuristic approach. Ultimately we are interested in the set of points that cross the moving viewplane between frames. This is the set of points lying in the double wedge defined by the two positions of the viewplane (at consecutive frames). These points correspond to polygons that switch from backfacing to frontfacing (or vice versa); all the silhouette changes are with edges adjacent to these polygons. We therefore need to answer double-wedge queries on a set of points (segment endpoints) in three dimensions, and then process the reported points.

Note that not all the points in the wedge contribute to changes in the silhouette. Suppose there is a segment completely contained in the double wedge that corresponds to

the current pair of frames. Then both facets adjacent to the primal entities of the endpoints of the segment have switched from being backfacing to frontfacing (or vice-versa), so the edge shared by the two facets does not become a silhouette edge. Thus we are interested only in segments that cross the wedge planes, and exactly one of whose endpoints is in the wedge.

Double-wedge point queries can be answered in $O(n^{2/3} + k)$ time, where n is the number of wedges and k is the output size. The algorithm is based on *partition trees* [24]. It is quite difficult to implement and inefficient in practice. (See a description of its implementation in [26, §3.4].) We have experimented with two search procedures for these queries: one based on the well-known *Octree*, and the other based on the *BAR* (balanced aspect ratio) tree of Duncan et al. [9]. In theory, The BAR tree combines the advantages of the k-d tree and the Octree. It has a small depth (logarithmic in the number of points) and at the same time the regions defined by the tree have bounded aspect ratio. Both characteristics are achieved by supporting diagonal cuts in addition to the axis-parallel cuts allowed by the traditional data structures. Our empirical results show that BAR trees, although superior in theory, do not perform as well as Octrees for data size smaller than roughly a million triangles. We have therefore used the Octree implementation in our experiments reported in Section 3.

The query-answering algorithm is straightforward for both Octree and BAR tree: we examine the regions of the tree, recursing only on those that intersect the query wedge. For each region we maintain:

1. A list L_{in} of dual points whose adjacent dual points are all contained in that region;

2. A list L_{out} of the rest of the points in the region; and

3. A pointer A to the region that contains all the dual points adjacent to the points in L_{out}.

For regions completely contained by the wedge, we do not report any points in L_{in}. At a leaf node we report all points in the wedge that have an adjacent point not in the wedge. Then, we reject in a second pass L_{out} points of each region whose A pointer is directed at a region out of the wedge. In theory, the running time of this procedure can be linear in the number of input points because we can construct an empty wedge (that does not contain any points) which intersects a linear number of regions. This case is however very rare. In practice, the algorithm performs very well and is easy to implement.

The tree-search procedure reports the dual points contained in the wedge, pruning out most of the useless points lying in the wedge. A (dual) point is useless if it is not adjacent to a silhouette. This happens if all its adjacent points also lie in the wedge. In theory, some of the reported points may yet have all adjacent points within the wedge. However, the fraction of such useless points is small in practice. Large solid models are often generally smooth. Smoothness implies that adjacent triangles have similar spatial orientations and hence their dual edges are small. As a result, most dual edges are short and are fully contained in the tree region containing their two endpoints. This effectively prevents most useless points from being reported by our algorithm.

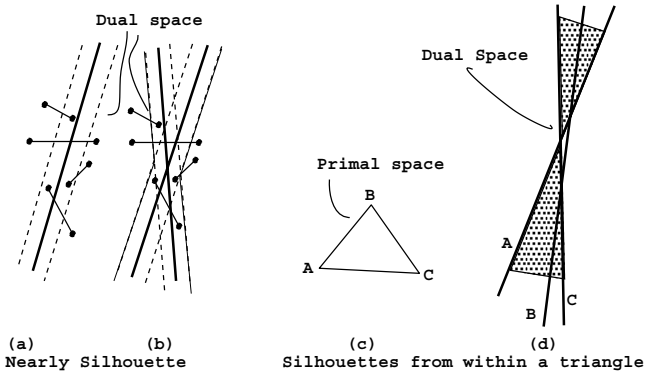


Figure 4: Silhouette extensions

The next step inspects each segment adjacent to the reported points. If the segment crosses one of the viewplanes, then its silhouette flag is toggled. Obviously, answering the query for the first time requires a brute-force computation of all the edges on the silhouette.

2.2 Extensions

2.2.1 Backface culling

Our silhouette-finding algorithm can also be used to compute the backfacing polygons. The duals of these polygons are points in the double wedge that change their backfacing status between frames. Points contained in one half of the wedge correspond to the polygons that change from being backfacing to frontfacing between the frames, while the points in the other half change from being frontfacing to backfacing. A backface-culling-only algorithm does not need, however, to answer wedge queries. A simple halfspace query results in the nodes that are frontfacing, which are then passed to the graphics hardware. The backfacing nodes are simply discarded. Leaf nodes intersecting with the viewplane require testing of each contained dual point. This backface-culling algorithm provides an average speed-up of about 50% over hardware backface culling for the bunny model. This compares favorably with earlier results [20, 31].

2.2.2 Near silhouettes

We can easily generalize our algorithm to find edges close to being on the silhouette (which is useful for model simplification). We demonstrate this in Figure 4. We know that silhouette edges cross the viewplane in the dual space. Edges close to being on the silhouette may be characterized as close to the viewplane, as shown in Figure 4(a), where we just consider two planes parallel to the viewplane. Figure 4(b) shows the region, bounded by solid thin lines, in which an incremental algorithm must locate the dual points for 'near-silhouettes.' Near-silhouette schemes are useful for applications with large overhead on silhouette update.

2.2.3 Generalized silhouettes

Some problems (e.g., shadow computation) require that we find edges that are on some silhouette seen from anywhere within a given triangle. Recall that all the silhouette updates along a line segment connecting two viewpoints are represented by dual points contained in the double wedge dual to the two viewpoints. Similarly, all updates associ-

ated with viewpoints within a triangle are given by the dual multi-wedge as shown by the shaded area in Figure 4(c,d): the region bounded by any pair of planes. A multi-wedge query may be answered just as a wedge query. We have to compute the intersection of all the wedges with the nodes of the Octree (or of the BAR tree).

3. EXPERIMENTAL RESULTS

We have implemented the entire algorithm in C; parts of it are also available in Java. The implementation took about 12 man months, and the software consisted of over 10,000 lines of code. All experiments were performed on an 64-bit SGI Octane workstation with one 195 MHz processor, 256 MB of internal memory, using an SSI graphics board, and running the IRIX 6.5.2m operating system. We have experimented with the algorithm on the models shown in Figure 5.

We first review the meaning of the observed parameters. The complexity of the models is presented through the number of vertices, facets, and edges. Another parameter of major importance in computer graphics is the *frame rate:* number of frames we can display per second. The parameters we examine are:

Edges/sil %: The percentage of silhouette edges out of the total number of edges. Intuitively, this parameter is proportional to the number of convex components of the model, since the number of silhouette edges of a convex polyhedron with n edges is roughly $\sqrt{n}$ [17].

Change/frame %: The percentage of silhouette edges that change between frames out of the total number of edges. The amount of work performed by our algorithm depends on this number, thus the smaller the number is, the more advantage we have over the brute-force approach.

Brute-force frame rate: Frame rate when computing the silhouette edges with the brute-force method, i.e., by examining all edges of the model and drawing each silhouette edge.

Our frame rate: Frame rate when computing the silhouette edges with our algorithm and drawing only the silhouette.

All-facets frame rate: Frame rate when displaying all the facets of the model (with hardware backface culling turned on).

All-edges frame rate: Frame rate when displaying all the edges of the model (no backface culling).

Our computation: Frame rate for silhouette computation using an Octree; rendering is turned off.

Brute-force Computation: Frame rate for silhouette computation using the brute-force method; rendering is turned off.

Our point-search procedure uses an Octree for smaller models and a BAR tree for larger models. For each model we ran several experiments set up as follows. We first created a sequence of viewpoints by randomly rotating, by hand (with a graphic interface), the bounding box of the model for 30 seconds, and recording the sequence of viewpoint positions. We then rendered the model in the modes listed above as it was viewed from for the recorded viewpoints.

Table 1 summarizes the characteristics of the models we tested. Although the percentage of silhouette edges of the model can be as high as about 30%, less than 1% of them changed between frames in our experiments. Thus the extra work we spent over the brute-force approach paid off.

Table 2 summarizes the differences between the brute-force approach and our algorithm, applied for some of the models. (Similar results were obtained for the other models.) We obtained a speed-up of more than 6 for larger models like the bunny and the dragon. For smaller models the rendering rate is limited by the graphics hardware buffer update rate. Observe that the brute-force approach is faster than our method for small models, which is the expected result.

Table 2 also provides a comparison of silhouette rendering time versus the rendering time of the (unsimplified) complete model, with hardware backface culling turned on. In all cases the frame rate is higher when drawing only silhouettes than when drawing all the edges or all the facets.

4. APPLICATIONS

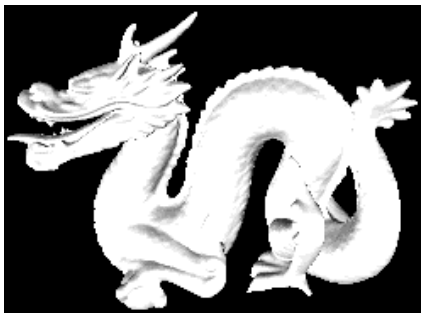
Efficient computation of silhouette edges is relevant for a variety of practical applications. In this section we briefly review a few of the problems which may benefit from our algorithm. (A more detailed discussion is given in the full version of the paper.)

4.1 Fast Rendering of Polyhedral Models

Computer-aided design (CAD) programs used in architecture and engineering require frequent manipulations of complex models. For such models, the image on the screen moves rather slowly due to the need to render the entire object during its motion. Visibility computation and model simplification are techniques commonly used to speed up rendering. A third approach is based on efficient silhouette computation. In this approach only the silhouette edges are rendered during quick movement and inspection. When the user stops moving the model, a more detailed rendering may be performed. Silhouettes capture the general structure of objects, and therefore can be used to guide the positioning of the object without the need to render all the edges. As an example, Figure 6 shows the silhouette of a bust of Beethoven as it is seen from three different viewpoints. The pose of the model can be easily inferred just from the silhouette.

4.2 Model Simplification

One of the central problems in computer graphics is that of *model simplification* [15, 16, 19, 22, 30]. The problem is to simplify the polyhedral representation of computer models so that the models preserve much of their original "look," yet use up much fewer facets for faster display. A general procedure for simplifying models involves a series of edge collapses such that the resulting model is not too far away (under an appropriate measure of distance) from the original model. The general framework for view-dependent simplification accounts for the viewpoint movement and simplifies the objects dynamically according to the viewpoint. The objects that come closer to the viewer need to show more detail and the objects that move away from the viewer can be simplified.

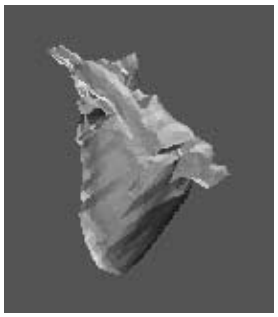


(a) Dragon

(b) Epcot

(c) Mushroom

(d) Bishop

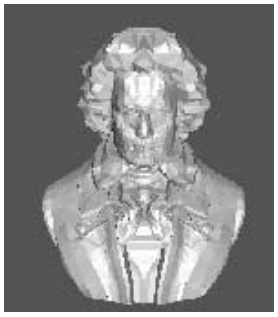


(e) Heart

(f) Head

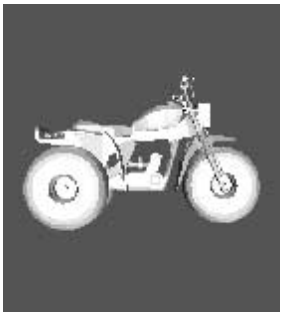
(g) Teapot

(h) Sky scraper

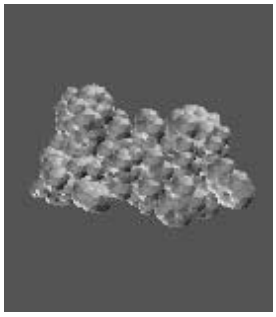


(i) Beethoven

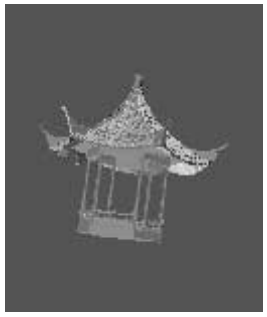
(j) Hammerhead



(k) Honda

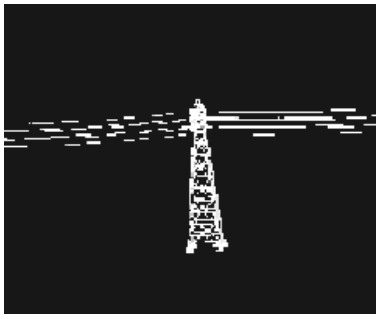


(l) Molecule



(m) Pagoda

(n) Bunny



(o) Power lines

Figure 5: Models with which we experimented

Dataset	Vertices	Facets	Edges	Edges/sil %	Change/frame %
bishop	251	496	744	12.76	0.48
heart	1,280	2,494	3,700	8.93	0.43
head	1,487	2,918	4,348	5.47	0.32
teapot	1,976	3,752	5,476	3.34	0.11
skyscraper	2,022	3,672	5,494	22.17	0.05
beethoven	2,521	5,030	7,545	11.97	0.32
hammerhead	2,560	5,116	7,674	6.74	0.17
honda	7,106	6,791	13,301	18.22	0.31
power lines	4,091	8,966	13,449	18.58	0.18
molecule	12,358	24,860	37,290	12.27	0.14
pagoda	24,254	40,098	57,960	21.85	0.28
bunny	34,834	69,451	104,065	3.09	0.10
dragon	437,645	871,414	1,309,257	2.9	0.05

Table 1: Complexities of test datasets

Dataset	Brute-force frame rate	Our frame rate	All-facets frame rate	All-edges frame rate	Our computation	Brute-force computation
bishop	72.29	71.52	72.31	72.26	1946.21	1566.3
teapot	71.20	71.88	71.63	72.07	941.51	522.32
skyscraper	70.31	71.35	59.81	70.30	922.58	261.22
molecule	24.55	42.78	12.32	30.59	94.69	17.11
pagoda	16.29	29.54	8.34	21.03	83.49	15.22
bunny	11.29	32.58	3.45	12.44	196.56	30.76
dragon	1.1	4.3	0.8	1.5	12.2	2.0

Table 2: A comparison between our approach and the brute-force silhouette computation

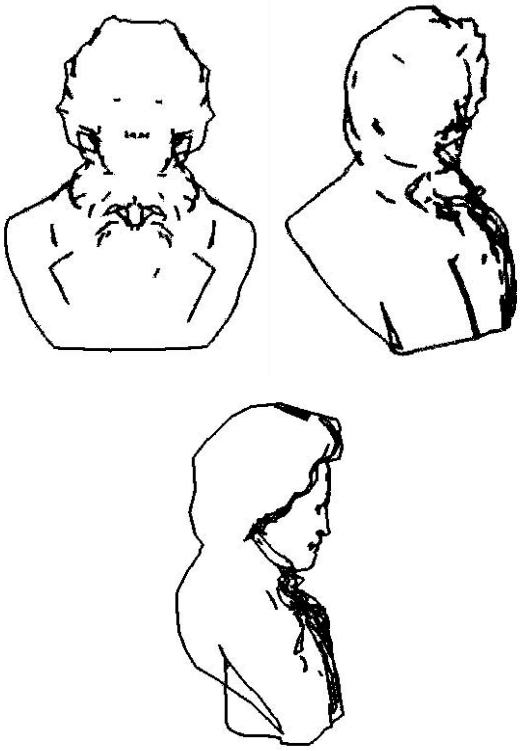


Figure 6: The silhouettes of Beethoven from three different viewpoints

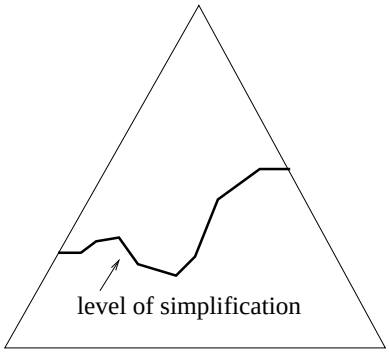
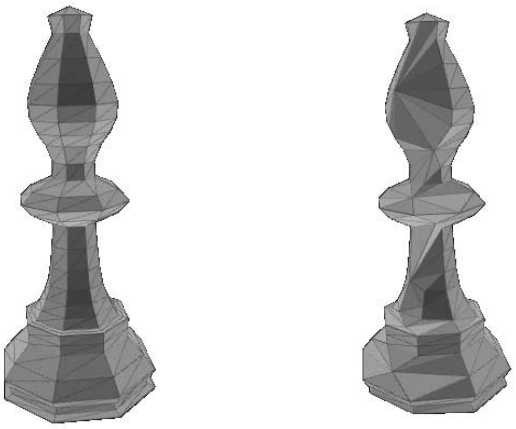


Figure 7: Progressive mesh tree

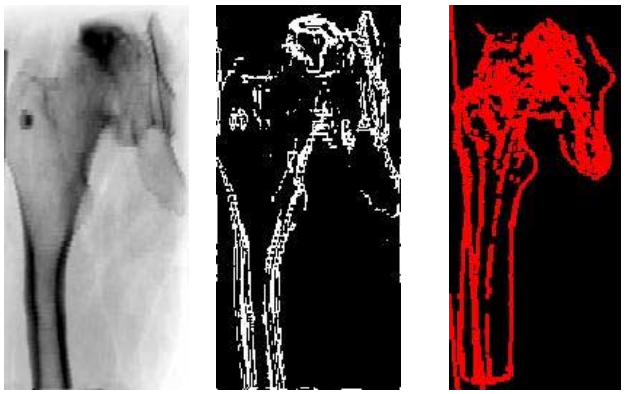
Fully dynamic computation of the adaptive simplification is time consuming. Therefore approaches like *progressive meshes* [15, 16] store a predetermined set of operations. The edges of the object are iteratively collapsed until the object becomes a tetrahedron. The history of collapses is stored in memory during a preprocessing step in the form of a tree. During display, the simplified model is obtained by examining a slice of the tree (see Figure 7). As the viewpoint moves, we move up and down in the tree and do or undo the collapses specified by the nodes of the tree.

Perspective-accurate silhouette edges may be used to guide the simplification process. We make sure that silhouette edges are not simplified too drastically. Thus, the object preserves its silhouette structure and still looks like the original object, while the interior of the model can be greatly simpli-



(a) Original (b) Simplified

Figure 8: Models of a chess bishop



(a) X-ray (b) X-ray edges (c) Silhouette edges

Figure 9: Registration of a femur model

fied. We also introduce a variant of Hoppe's approach [15, 16] in the sense that the simplification tree is dynamically constructed and updated. We store only the part of the tree that is below the slice corresponding to the current simplification level. Essentially we store a history of all edge collapses performed to attain the current view. We can afford a dynamic edge-collapsing process as the surface-quality metric is trivial to compute. As an example of our approach, Figure 8 shows a chess bishop and a simplified version of it. The simplified model looks much like a bishop even when the interior of the silhouette is simplified quite aggressively.

4.3 Medical Image Registration

One of the important problems in medical robotics is that of pose registration between a detailed preoperative three dimensional model and X-rays taken during the operation. Some of the algorithms used for the registration [21, §7, pp. 115–143] involve the matching of features present in both the model and the X-ray images. Since silhouettes represent discontinuities in the visibility of the object, they form a subset of the features present in the X-rays and hence can be used to guide the registration algorithm. Figure 9 shows an X-ray of a femur, the edges in the image (as determined by standard image-processing techniques), and the silhouette of a three dimensional model of the femur. Similarities between the edges in the X-ray and the silhouette edges are visually obvious. We use a measure of similarity based on the distance between each silhouette point on the image and the closest edge on the X-ray image.

Figure 10 shows the same three pictures of a model of a brain. Note that in this case some of the X-ray features are not captured by the silhouette. This is due to the fact that X-rays also capture features effected by density gradients. Such features cannot be described by the polyhedral models we use. We can therefore use a silhouette-based method for obtaining fast an approximate solution to the registration problem, and apply then a more laborious technique for arriving closer to the correct solution.

5. CONCLUSIONS

We present a nontrivial algorithm for computing true silhouette edges in polyhedral models under perspective projection. Our algorithm is not only theoretically efficient, but it also performs well in practice. We provide experimental data that show the efficiency of our algorithm with respect to prior approaches to silhouette computation. We also present a set of practical applications that use our algorithm. We are continuing further research in these application areas to make the best use of silhouettes. We are also pursuing techniques to compute only the extremal silhouettes—the silhouette edges that are not enclosed on-screen by other edges. Extremal silhouettes are useful, for example, for shadow computation. In addition, we are working on a mechanism which will automatically decide whether to use an Octree or a BAR tree for the point-location operations.

6. ACKNOWLEDGEMENTS

The authors wish to thank numerous individuals and institutions who contributed the datasets used in this research, most notably Lutz Kettner from ETH Zürich, the Stanford graphics group, and ERC, the NSF Engineering Research Center on computer-assisted systems and technology.

7. REFERENCES

[1] P.K. Agarwal and M. Sharir, Applications of a new space-partitioning technique, *Discrete & Computational Geometry*, 9 (1993), 11–38.

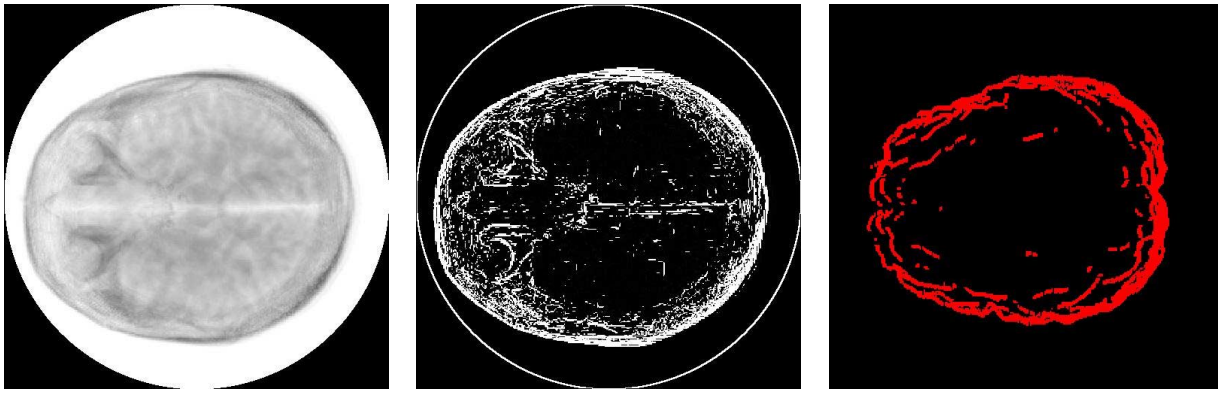
[2] G. Barequet, C.A. Duncan, M.T. Goodrich, W. Huang, S. Kumar, and M. Pop, Efficient perspective-accurate silhouette computation, *Proc. 4th CGC workshop on Computational Geometry*, Baltimore, MD, 1999.

[3] F. Benichou and G. Elber, Output sensitive extraction of silhouettes from polygonal geometry, *Proc. 7th Pacific Graphics Conference*, Seoul, Korea, 60–69, 1999.

[4] W. Bouma and G. Vanecek, Velocity-based collision detection, in *Graphics Gems V* (A. Paeth, ed.), 380–385, Academic Press, 1995.

[5] J.W. Buchanan and M.C. Sousa, The edge buffer: A data structure for easy silhouette rendering, *Proc. 1st Int. Symp. on Non Photorealistic Animation and Rendering*, Annecy, France, 39–42, 2000.

[6] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, *SIAM Journal of Computing*, 17 (1988), 427–462.



(a) X-ray (b) X-ray edges (c) Silhouette edges

Figure 10: Registration of a brain model

[7] B. Chazelle, M. Sharir, and E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems, *Algorithmica*, 8 (1992), 407–429.

[8] F. Crow, Shadow algorithms for computer graphics, *ACM Computer Graphics*, 11 (1977), 342–248.

[9] C.A. Duncan, M.T. Goodrich, and S. Kobourov, Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees, *J. of Algorithms*, 38 (2001), 303–333.

[10] H. Edelsbrunner, M.H. Overmars, and R. Seidel, Some methods of computational geometry applied to computer graphics, *Computer vision, graphics, and image processing*, 28 (1984), 92–108.

[11] A. Efrat, L.J. Guibas, and O.A. Hall-Holt and L. Zhang, On incremental rendering of silhouette maps of a polyhedral scene, *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, CA, 910–917, 2000.

[12] B. Gooch, P.J. Sloan, A. Gooch, P. Shirley, and R. Riesenfeld, Interactive technical illustration, *Proc. ACM Symp. on 3D Interactive Graphics*, Atlanta, GA, 31–38, 1999.

[13] G. Gornowicz and L. Williams, Snap to it! Automatic 3D object and silhouette registration, *Proc. ACM SIGGRAPH Technical Sketch*, 258, 2000.

[14] A. Hertzmann and D. Zorin, Illustrating smooth surfaces, *Proc. ACM SIGGRAPH*, 517–526, 2000.

[15] H. Hoppe, View dependent refinement of progressive meshes, *Proc. ACM SIGGRAPH*, 189–198, 1997.

[16] H. Hoppe, Efficient implementation of progressive meshes, Technical Report MSR-TR-98-02, Microsoft Research, 1998.

[17] L. Kettner and E. Welzl, Contour edge analysis for polyhedron projections, in *Geometric Modeling: Theory and Practice* (W. Strasser, R. Klein, and R. Rau, eds.), 379–394, Springer-Verlag, 1997.

[18] S.K. Kim, Parallel algorithms for the segment dragging problem, *Information Processing Letters*, 36 (1990), 323–327.

[19] L. Kobbelt, S. Campagna, and H.P. Seidel, A general framework for mesh decimation, *Proc. of Graphics Interface*, 43–50, 1998.

[20] S. Kumar, D. Manocha, W. Garrett, and M. Lin, Hierarchical back-face culling, *Computers and Graphics*, 23 (1999), 681–692.

[21] S. Lavallée, R. Szeliski, and L. Brunie, Anatomy-based registration of three-dimensional medical images, range images, X-ray projections, and three-dimensional models using octree-splines, in *Computer-Integrated Surgery* (R.H. Taylor, S. Lavallée, G.C. Burdea, and R. Mösges, eds.), MIT Press, 1995.

[22] D. Luebke and C. Erikson, View-dependent simplification of arbitrary polygonal environments, *Proc. ACM SIGGRAPH*, 199–208, 1997.

[23] L. Markosian, M. Kowalski, S. Trychin, L. Bourdev, D. Goldstein, and J. Hughes, Real-time nonphotorealistic rendering, *Proc. ACM SIGGRAPH*, 415–420, 1997.

[24] J. Matoušek, Range searching with efficient hierarchical cuttings, *Discrete & Computational Geometry*, 10 (1993), 157–182.

[25] W. Matusik, C. Buehler, S. Gortler, R. Raskar, and L. McMillan, Image based visual hulls, *Proc. ACM SIGGRAPH*, 369–374, 2000.

[26] M. Pop, Exploiting coherence in spatial database queries, Ph.D. Thesis, Johns Hopkins University, Baltimore, MD, 2000.

[27] F.P. Preparata and M.I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[28] R. Raskar and M. Cohen, Image precision silhouette edges, *Proc. Symp. on Interactive 3D Graphics*, Atlanta, GA, 135–140, 1999.

[29] P. Sander, S. Gortler, H. Hoppe, and J. Snyder, Silhouette clipping, *Proc. ACM SIGGRAPH*, 327–334, 2000.

[30] J.C. Xia and A. Varshney, Dynamic view-dependent simplification for polygonal models, *Proc. IEEE Conf. on Visualization*, 327–334, 1996.

[31] H. Zhang and K. Hoff, Backface culling using normal masks, *Proc. Symp. on Interactive 3D Graphics*, Providence, RI, 103–106, 1997.