EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos
(Guest Editors)

*Volume 25* (*2006*), *Number 3*

# Silhouette Extraction in Hough Space

Matt Olson and Hao Zhang †

GrUVi lab, School of Computing Science, Simon Fraser University, Burnaby, Canada

**Abstract**

*Object-space silhouette extraction is an important problem in fields ranging from non-photorealistic computer graphics to medical robotics. We present an efficient silhouette extractor for triangle meshes under perspective projection and make three contributions. First, we describe a novel application of 3D Hough transforms, which allows us to organize mesh data more effectively for silhouette computations than the traditional dual transform. Next, we introduce an incremental silhouette update algorithm which operates on an octree augmented with neighbour information and optimized for efficient low-level traversal. Finally, we present a method for initial extraction of silhouette, using the same data structure, whose performance is linear in the size of the extracted silhouette. We demonstrate significant performance improvements given by our approach over the current state of the art.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Visible line/surface algorithms

## 1. Introduction

Silhouette extraction is a fundamental problem in computer graphics, computational geometry, and related fields. Some of its applications include non-photorealistic rendering and animation [HZ00, IHS02, MKG*97], volume visualization [BKR*05], occlusion volume generation [Cro77], geometry reduction [SGG*00], volumetric lighting and shadowing, and object tracking in computer vision [PF02]. Many of these problems involve scene rendering or analysis under perspective projection. Silhouette extraction in perspective is more challenging than the case of parallel projection, as the view vector is not constant over the scene.

Two main approaches for silhouette extraction have been developed, which carry out similar tasks but in fundamentally different ways. One approach works in the *image space* and detects discontinuities in a rendered image, e.g., a depth buffer. The other works in the *object space* where edges separating front- and back-facing regions of a polygonal mesh are explicitly identified. Image-space algorithms are generally quite fast and easily accelerated using graphics hardware, but can only identify visible portions of a model's silhouette loops, rendering these algorithms less useful for applications such as shadow volume extrusion [Cro77]. In addition, image-space algorithms can only produce silhouettes to image precision. By contrast, object-space algorithms work in floating-point precision and can generate a more broadly applicable set of edges which can be subsequently stylized [IHS02]. However, object-space algorithms tend to be slower than their image-space counterparts and are not as amenable for hardware implementations.

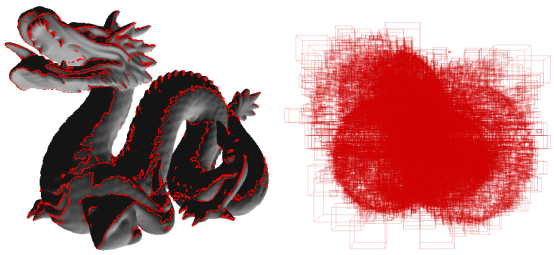


**Figure 1:** *Extracted silhouette (red edges) on the dragon mesh, with respect to a light source to the lower right, and a visualization of the Hough transform of the mesh triangles, showing bounding boxes for low-level octree nodes.*

Object-space methods can be further divided into *brute-force* and *output-based* algorithms. Brute-force algorithms, e.g., edge buffer [BS00], must check every face and edge in

† mjolson@cs.sfu.ca, haoz@cs.sfu.ca

the mesh at every frame; output-based algorithms [HZ00, SGG*00, PDB*01] aim to minimize the number of non-silhouette edges examined (we are not aware of any provably output-sensitive object-space silhouette algorithm under perspective projection). We explore this distinction, along with existing silhouette extraction algorithms, in Section 2.

In this paper, we present an object-space, output-based silhouette extraction algorithm. We structure our data in the 3D Hough space, which is shown to possess more desirable properties than the often used dual space [PDB*01]. Our spatial search data structure efficiently supports both incremental silhouette update — calculating changes in the silhouette set from frame to frame — and static silhouette computation — applicable at the start of an animation or after a large camera movement. The main motivation for our work is that in a typical polygonal environment, the size of the silhouette set is much smaller than the size of the scene [KW97] and silhouette change between consecutive frames is even smaller. Our main contributions are:

- A novel application of the 3D Hough transform to object-space silhouette extraction. The more favourable point distributions obtained, compared to those resulting from applying the dual-space transform, already give a significant speed-up in silhouette updates, with no other changes to the existing algorithm of Pop et al. [PDB*01].
- A new silhouette update algorithm based on an incremental low-level traversal of a novel search graph, which augments our octree data structure, provides further performance improvement.
- A simple characterization of mesh edges with respect to the origin allows us to achieve efficient (experimentally shown to be output-sensitive) initial extraction of silhouettes using the same octree-based data structure.

The remainder of the paper is organized as follows. Section 2 outlines previous work, focusing on object-space approaches. Next, we give a theoretical treatment of 3D Hough transform and discuss its application to silhouette computation. We also give an overview of our data structure and algorithms, with a more detailed coverage provided in Sections 5 and 6. In Section 7, we demonstrate experimentally that our approach outperforms the current state of the art. Finally, we conclude in Section 8 and suggest future work.

## 2. Previous work

Image-space silhouette algorithms exploit discontinuities in the image buffers obtained from controlled scene rendering and detect the discontinuities via image processing, e.g., edge detection [IFH*03]. They are fast, hardware-friendly, requiring no expensive preprocessing, and even applicable for dynamic scenes [RC99]. However, these algorithms do not return an analytical representation of the silhouette set. In this section, we overview object-space approaches and refer those interested in image-space and hybrid algorithms to the excellent survey of Isenberg et al. [IFH*03].

The most obvious method for finding the silhouette edges of a mesh is simply to consider every edge, testing its adjacent faces and adding it to the silhouette if one is front-facing and the other is back-facing. Buchanan and Sousa [BS00] avoid checking each face three times by using the *edge buffer*, which stores front- and back-facing flags for each edge; these are set as each face is rendered. Silhouette extraction involves traversing the edge buffer and examining the flags. However, the work done is proportional to the size of the whole model and not to the size of the output.

Empirical studies by Kettner and Welzl [KW97] show that for most commonly-used mesh models, about 10% of the edges appear on the silhouette. Thus an output-sensitive silhouette algorithm, despite of a higher cost per face/edge tested, can be quite desirable. For orthographic projections, Benichou and Elber [BE99] propose an elegant algorithm utilizing a central projection of the Gaussian image onto a tight bounding cube. Based on a regular subdivision of the cube sides, they develop an approximately output-sensitive silhouette extraction algorithm by enumerating all projected arcs on the cube that intersect the view plane. This approach depends on the view-vector/view-plane duality, which only holds under orthographic projection. Gooch et al. [GSG*99] also rely on the Gaussian sphere, but use a hierarchy of spherical triangles to facilitate their search.

Hertzmann and Zorin [HZ00] extend the above dual-space approach into 4D homogeneous space to find silhouettes under perspective projection. The problem is turned into one of intersecting the triangles on a hypercube's sides with the view point's dual plane, which is conceptually more difficult than that of finding line-line intersections on the planar sides of a 3D cube. However, an octree-based spatial subdivision can speed up silhouette search. From sparse empirical data, it is reported that the algorithm's performance "is roughly linear in the number of silhouette triangles." Another hierarchical search structure adopted for silhouette computation is the spatialized normal cone hierarchy [JC01].

Pop et al. [PDB*01] also use a dual-space approach for silhouette extraction in the perspective, but stay in 3D; we describe this dual-space transform in Section 4. They only solve the incremental silhouette update problem, using an octree of points that are dual to the mesh faces. They note that the view point at two frames correspond to two intersecting view planes in dual space and only points lying in the two wedges formed by the planes can incur a silhouette change. The authors perform range-search queries against the wedges at each frame by always traversing the octree from its root. They also point out that standard octrees tend to perform better than more theoretically optimal search structures — such as bounded aspect ratio trees [DGK01] — for meshes with less than a million triangles.

Instead of using a search tree based on spatial subdivisions [BE99, HZ00, PDB*01], Sander et al. [SGG*00] construct a hierarchical search forest based on a discrete cluster-

ing of mesh edges. Each node in an edge tree stores a set of edges and two anchored cones as conservative estimates of the front- and back-facing regions with respect to all faces incident to edges stored at the node and to edges stored in its subtrees. If the view point is inside either cone, the node and its children are culled. Otherwise, each edge associated directly with the node must be tested, and the node's children are processed recursively. A costly optimization is performed in preprocessing to cluster the edges, forming the search forest, so as to minimize an expected silhouette extraction cost. Performance roughly linear in the size of the extracted silhouette is reported. But since spatial coherence is mostly lost in the edge trees, they cannot be utilized to perform frame-to-frame silhouette updates.

In this paper, we advocate the use of the 3D Hough transform of mesh faces, organized into an augmented octree, for efficient update and initial extraction of silhouette. 2D Hough transforms are well known constructs for line and circle detections in an image [Jai89]. In a similar spirit, 3D Hough transforms have been used for mesh retrieval [ZP01]. The dual-space transform has often been used for silhouette [PDB*01] and other visibility computations, e.g., back face culling [KMGL96]. Although the 3D Hough transform has a simple connection to the dual transform (See Section 4.1), to the best of our knowledge, it has not been considered for the same type of problems before. The only other application of 3D Hough transform to mesh processing we are aware of is the work of Décoret et al. [DDSD03] and Meseth and Klein [MK04], for billboard clouds rendering, where the planes in a scene are mapped into Hough space in which their density is estimated via binning.

## 3. Overview of data structure and algorithms

We begin with a set of triangle meshes in a scene and transform each triangle into a point in 3D Hough space. The backbone of our search data structure is an octree built upon these Hough-space points. To facilitate fast silhouette extraction, each octree node, in addition to storing a set of Hough-space points spatially contained within the octree node, is augmented with two bounding volumes.

- **The point bounding volume (PBV):** The PBV is the tightest bounding volume of the set of Hough-space points belonging to the octree node. It can be utilized for more effective culling due to its tighter bounding of the set of Hough-space points in the octree node.
- **The edge bounding volume (EBV):** The EBV is the tightest bounding volume of the set of Hough-space points *related* to one or more Hough-space points in the octree node. We shall make clear of this relation between the mesh faces later on when we describe our algorithm. The EBVs are present so that trivial accept or reject of mesh edges against the silhouette set can be facilitated.

In 3D hough space, the view point is mapped to a sphere, referred to as the *v-sphere*, passing through the origin and the view point itself. The majority of silhouette edges, with respect to the view point, correspond exactly to the set of Hough-space point pairs, in which one point lies inside the v-sphere and the other lies outside. Those that do not fall into this category are edges that are on the silhouette with respect to the origin. For initial silhouette extraction, we apply a standard top-down, hierarchical octree-based culling scheme using the v-sphere. All octree nodes outside the v-sphere can be immediately culled away. Any octree node whose EBV is entirely contained in the v-sphere can also be culled. Recursion stops when the size of the octree node reached, measured by the number of Hough-space points it contains, is sufficiently small. At this point, mesh edges relevant to these points are tested to see whether they are silhouette edges. In addition, all edges that are on the silhouette with respect to the origin are also tested explicitly.

In many interactive applications, e.g., virtual walkthrough or object tracking, the ability to quickly update the silhouette is highly desirable. Pop et al. [PDB*01] keep track of the difference between the silhouette sets at consecutive frames, in the dual space, and always start the search at the root of the octree. We show that with the same algorithm, but executed in Hough space, the performance gain is already substantial. We improve performance further by taking advantage of frame-to-frame coherence and conduct an incremental neighbour search through the octree. A novel neighbour graph augments our octree and an appropriate selection of search nodes allows us to keep track of silhouette changes efficiently. Specifically, we only need to examine octree nodes that straddle or are contained in the *active region* defined by the v-spheres at two consecutive frames.

## 4. 3D Hough transform for silhouette extraction

In this section, we provide the mathematical background on 3D Hough transforms and show how they can be applied effectively for silhouette computation. We also relate the Hough transform to the well-known dual-space transform and demonstrate the advantages offered by the former.

### 4.1. 3D Hough transform and dual-space transform

Given a plane $\pi : ax+by+cz+d=0$ with normal $(a,b,c)$ having unit length, if $\pi$ does not pass through the origin, i.e., $d \neq 0$, then its *dual*, denoted by $\mathscr{D}(\pi)$, is the 3D point $(-a/d,-b/d,-c/d)$; otherwise, $\mathscr{D}(\pi)$ is the *point at infinity*. Conversely, given any 3D point $(a',b',c')$, its *dual plane* is given by $a'x+b'y+c'z=1$. The 3D Hough transform $\mathscr{H}(\pi)$ of the plane $\pi$ is the 3D point $(-ad,-bd,-cd)$ and there are no constraints placed on any of the coefficients.

Geometrically, the 3D Hough transform of a triangle $T$ can be constructed by drawing a line, from the origin, that is perpendicular to the support plane of the triangle. The point of intersection $H$, between the line and the support plane, is defined to be the 3D Hough transform of the triangle, as

shown in Figure 2. All triangles sharing the same support plane, even those with opposite orientations, are mapped to the same point in Hough space.

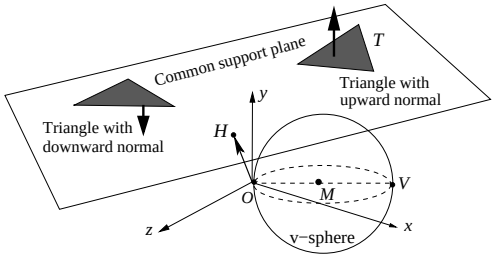



**Figure 2:** *3D Hough transform H of a triangle T and the sphere (v-sphere) corresponding to the view point V.*

**Relating Hough and dual spaces via inversion:** An elegant way to relate the Hough transform to the dual transform is through the notion of *inversion* with respect to the unit sphere centered at the origin. In general, points $P$ and $P'$ are said to be inverses of each other with respect to a sphere of radius $r$ and centered at $O$, called the center of inversion, if $P'$ lies on the ray $\overrightarrow{OP}$ and $|OP| \cdot |OP'| = r^2$. By definition, the point at infinity is inverted to the origin. Inversion in a circle or sphere is a well-studied geometric transformation [Bla00]. Among its many interesting properties, e.g., conformality, we point out one that is relevant to our discussion. Namely, the inverse of a line (plane) not passing through the center of inversion is a circle (sphere) passing through the center of inversion, and vice versa. It follows that the dual of a view point $V$, which is a plane not through the origin, is mapped via inversion to a sphere passing through the origin $O$ in Hough space, as shown in Figure 2. It can be shown that this sphere, which we call a *v-sphere*, is centered at the midpoint $M$ of the line segment $OV$.

**Advantages of Hough transforms:** It is easy to show that any bounding sphere $S$, centered at the origin, of a set of triangles also bounds the Hough transforms of the triangles. The tightest bounding sphere for the dual-space transforms however can be much larger than $S$. Inversion causes dual-space points to exhibit a highly nonuniform distribution, typically with severe clustering about the origin as well as points extremely far away from the origin; see Figure 3 for an example. Besides the precision issues, which influence the numerical stability of the algorithms, octrees constructed in dual space tend to have high and unevenly distributed leaf depths. These lead to poor performance in both initial and incremental silhouette extraction, compared to the use of Hough transforms. Note that Hough-space transforms of mesh models would also exhibit some level of clustering around the origin. But empirically, Hough-space silhouette extraction shows far superior performance than its dual-space counterpart, with the same data structure construction and search algorithm, e.g., as given in Pop et al. [PDB*01].

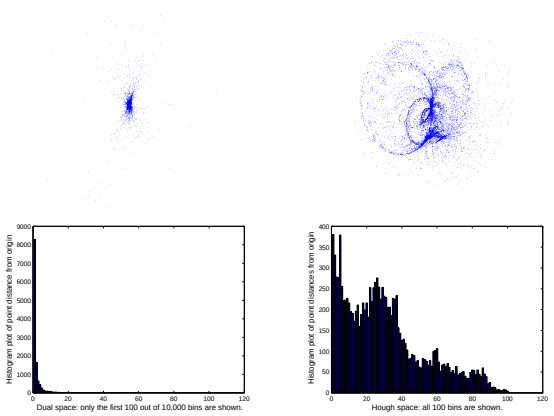



**Figure 3:** *Scattered point plots (top) and histogram plots of distances from the origin (bottom) for the hand model shown in Figure 8. Left: dual space. Right: Hough space. The origin is chosen as the centroid of the model. Point plot in dual space shown is obtained after 10 levels of zooming in Matlab, while the Hough-space plot is shown as is. Some dual-space points are extremely far from the origin and not visible in the figure. Evidently, point distribution in Hough space is much more uniform (less clustering around the origin). This example is representative of the general trend.*

### 4.2. Silhouette computation in Hough space

Given two adjacent triangles $T_1$ and $T_2$ sharing a common edge $e$ and with respective support planes $\pi_1$ and $\pi_2$, edge $e$ is a silhouette edge with respect to a view point $V$ if and only if the plane $\pi_1$, as it rotates about $e$ until reaching $\pi_2$, would hit $V$; refer to Figure 4(a). Note that the angle of rotation here needs to be less than $\pi$. In Hough space, the rotating plane would trace out a *circular arc*, whose end points are the Hough transform $H_1 = \mathscr{H}(\pi_1)$ and $H_2 = \mathscr{H}(\pi_2)$, as shown in Figure 4(b). The full circle, which we call the *Hough circle* for $e$, is defined by a diameter whose end points are $O$, the origin, and $E$, the intersection between the line extension of $e$ and a plane passing through $O$ and perpendicular to $e$. We define the Hough transform of the mesh edge $e$ to be this circular arc traced out by the rotating plane. Clearly, the orientation of the triangles incident to $e$ determine whether the arc contains the origin; this is precisely related to whether $e$ is a silhouette edge, when viewed from the origin $O$.

**Theorem 1** : The Hough transform of an edge $e$ contains the origin $O$ if and only if $e$ is a silhouette edge, viewed from $O$.

**Theorem 2** : Assume that no edges or their line extensions pass through the origin. Then an edge $e$ is a silhouette edge with respect to a view point $V$ if and only if the Hough transform of $e$ is tangent to the v-sphere defined by $V$ or it intersects the v-sphere through a point other than the origin.

Theorems 1 and 2 are the Hough-space equivalents of Theorems 1 and 2 from Pop et al. [PDB*01]. Figure 5 de-

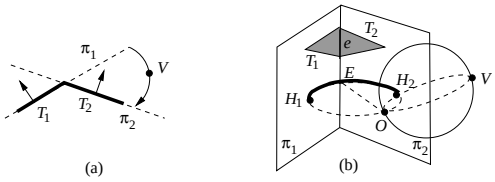



**Figure 4:** *The Hough transform of an edge. (a) Plane rotation hits view point. (b) Plane rotation traces out a circular arc (thickened arc between $H_1$ and $H_2$), defined as the Hough transform of the edge e incident to triangles $T_1$ and $T_2$.*

picts the different cases, in an orthographically projected view with projectors parallel to the edge in question, as an illustration for Theorem 1. The assumption in Theorem 2 can be ensured through perturbation in preprocessing. An elementary proof of Theorem 2, one without resorting to the use of inversion, is given in the Appendix.

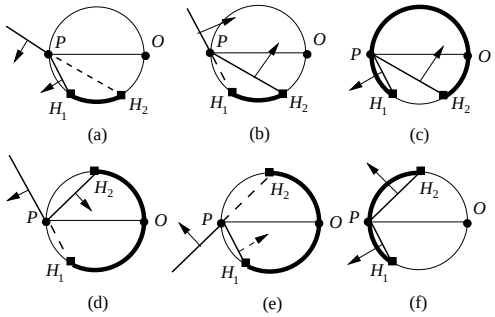



**Figure 5:** *The Hough transform (thickened arc) of an edge, projected to P, contains the origin O if and only if it is a silhouette edge, viewed from O; arrows depict plane normals.*

**Corollary 1:** If $e$ is not a silhouette edge, viewed from the origin, then it is a silhouette edge with respect to a view point $V$ if and only if $H_1$ and $H_2$, the Hough transform of faces incident to $e$, lie on opposite sides of the v-sphere associated with $V$. If $e$ is a silhouette edge, viewed from the origin, then it is a silhouette edge with respect to $V$ if and only if $H_1$ and $H_2$ lie on the same side of the v-sphere.

This corollary allows us to speed up initial silhouette extraction, since the majority of the edges are typically not on the silhouette [KW97], when viewed from the origin. To extract silhouette from this set of edges, with respect to a v-sphere, one only needs to examine octree nodes inside or intersecting the v-sphere — details are given in Section 6.1.

The situation for incremental silhouette updates is simpler, as we are interested only in the *change* in the silhouette set. From frame $t$ to frame $t+1$, the membership of an edge with respect to the silhouette set changes if and only if the front- or back-facing status of one of its incident faces changes with respect to the moving view point. Thus it is sufficient to examine octree nodes inside or intersecting the active region between the v-sphere at the two frames.

## 5. Augmented octree in Hough space

We use a bounded augmented octree, built on top of the set of Hough-space points corresponding to a set of triangles in a scene, for silhouette extraction. The root of the octree represents the tightest, axis-aligned bounding box of the whole point set. Each node in the octree stores the following.

- Eight pointers to its children. A node containing no Hough-space points is assigned to be `NULL`.
- Six pointers, possibly `NULL`, to its neighbour nodes; this is explained in detail in Section 5.1.
- Three axis-aligned bounding volumes, each requiring six floats. In addition to the **OBV**, which bounds the full octant associated with the node, we also store
  - **PBV:** the tightest bounding box of all the Hough-space points enclosed by the node; let us denote this set by $W$ for now. Note that PBV is enclosed by OBV.
  - **EBV:** the tightest bounding box of the set of Hough-space points *related* to a point in $W$. Specifically, a Hough-space point $H$ is related to $H' \in W$ if $H = H'$ or the triangles corresponding to $H$ and $H'$ share an edge and this edge is *not* on the silhouette when viewed from the origin. The EBV is utilized for static silhouette extraction, as explained in Section 3 and 6.1.
- **Extra data** to indicate whether the node makes a good candidate for the *active set*. The active-set candidates determine where in the octree we perform neighbour traversal; this is described in details in Section 6.3.

In our current implementation, octree nodes are recursively subdivided until each non-empty leaf node has precisely one Hough-space point. In general, one can stop the subdivision when the number of Hough-space points in a node falls below a user-defined threshold.

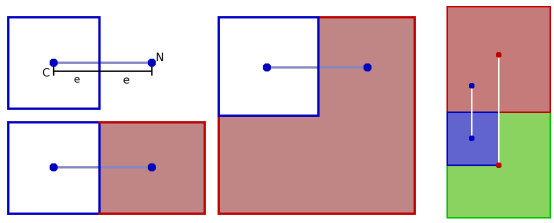
### 5.1. The neighbour graph

During incremental silhouette update, we walk from properly selected octree nodes to their neighbours rather than always searching from the tree's root, as in [PDB*01]. We build a *directed neighbour graph* whose vertices are the octree nodes and whose edges connect a node to its neighbours. A node may have up to six neighbours, one across each face of its OBV. Nodes on the boundary of the octree do not have neighbours along those boundary faces.

Since our octrees are not fully populated in general, it is not always immediately obvious which nodes are neighbours of a given octree node. To determine a node's neighbour in a given direction $\vec{u}$ (there are six such directions as given above), where $\vec{u}$ is of unit length, we first find a *neighbour point*. Consider an octree node whose OBV is centered at

point $C$ with extent or half-width $e$ along the direction of $\vec{u}$. The node's neighbour point $N$ with respect to $\vec{u}$ is $C + 2e\vec{u}$, as shown in the upper-left diagram of Figure 6(a).

We define the *neighbour node* of an octree node $R$, in a given direction, as the deepest node in the tree, no deeper than $R$, that contains $R$'s neighbour point in that direction. Typically, $R$'s neighbour would be at the same depth as $R$, as shown in the lower-left diagram of Figure 6(a). But this is not guaranteed if the octree is not full. For example, a node's neighbour may be its parent, as shown on the right side of Figure 6(a). It is also worth noting that the neighbour relation is not symmetric in general, as shown in Figure 6(b). Symmetry is ensured only between two neighbouring nodes that are at the same depth in the octree.




(a) (b)

**Figure 6:** *2D illustrations of the neighbour relations. (a) Upper-left: an octree node centered at C with its east neighbour point N. Lower-left: two neighbouring nodes at the same depth. Right: the east neighbour of a node is its parent. (b) Asymmetry of the neighbour relation: the north neighbour of the blue node is the red node, whose south neighbour is the green node (parent of the blue node).*

### 5.2. Edge list

In addition to the augmented octree, we maintain an edge list to store relevant information for each edge: the position of each vertex on the edge, and facing (front or back with respect to the current view point) flags for each face adjacent to the edge. We can therefore easily determine whether an edge is on the silhouette or not by checking the facing flags of its incident faces, as done for edge buffers [BS00].

## 6. Silhouette extraction algorithms

In Section 6.1, we describe our initial silhouette extraction scheme. Subsequently, we offer two options for identifying changes to the silhouette on a frame-by-frame basis. One involves a full traversal of the Hough-space octree from the root at every frame; the other maintains a list of active octree nodes near the silhouette and proceeds incrementally. Both algorithms work on the same principle. We are only concerned with faces that have passed from front-facing to back-facing, or vice versa. A face changes its facing status when it *crosses* the v-sphere (defined in Section 3). Thus any Hough-space point that has changed its facing status must lie within the symmetric difference between the v-spheres at the current and the previous frames; we refer to this symmetric set difference as the *active region*.

We maintain two v-spheres, one for the current frame and one for the previous frame, during view point changes. At each frame, we consider octree nodes whose associated PBVs intersect the active region. Once we have identified all the faces whose facing status relative to the view point has changed, we change the edge flags associated with these faces, retest those edges affected for silhouette status, and add or remove them from the silhouette set of the scene.

### 6.1. Initial silhouette extraction

In preprocessing, we determine the set $\mathscr{S}^+(O)$ of edges that are on the silhouette with respect to the origin $O$. Denote the remaining set of edges by $\mathscr{S}^-(O)$. Given a view point $V$, all edges in $\mathscr{S}^+(O)$ will simply be tested explicitly against $V$ to determine their silhouette status with respect to $V$. Note that $|\mathscr{S}^+(O)|$ is expected to be small and comparable to the size of the silhouette set with respect to the view point $V$. Thus the cost of these explicit tests is expected to be roughly the same as the size of the extracted silhouette.

To extract silhouette edges with respect to $V$ from the set $\mathscr{S}^-(O)$, we traverse the octree from its root. Pseudocode for the recursive search algorithm is given below:

```
extract_initial_sil(node* n) {
   if (n == NULL || outside_vsphere(n->PBV)
                 || inside_vsphere(n->EBV))
      return;

   if (num_hough_pts_in_node(n) < threshold) {
      test_all_faces_in(n);
      return;
   }

   for (k = 0; k < 8; k++)
      extract_initial_sil(n->kids[k]);
}

test_all_faces_in(node* n) {
   for (f = 0; f < n->n_faces; f++) {
      determine_facing_of(n->faces[f]);
      check_edges_on(n->faces[f]);
   }
}
```

To initiate incremental silhouette search, to be described in Section 6.3, we add appropriate octree nodes that intersect the v-sphere at the *first time frame* to its *active set*. Details on active sets are given in Sections 6.3 and 6.4.

### 6.2. Full-traversal silhouette update

For full traversal, we start at the root of the Hough-space octree at each frame, analogous to Pop et al. [PDB*01], and descend into nodes whose PBVs intersect the active region. Recursion stops when the number of Hough-space points in a node is sufficiently small; these points are tested for facing to decide the silhouette status of their adjacent edges. If a node's PBV is contained by the active region, all the Hough-space points therein also undergo the facing test.

### 6.3. Incremental silhouette update

Between consecutive time frames $t$ and $t+1$, our incremental silhouette search aims at quickly identifying all Hough-space points lying within the active region; mesh faces corresponding to these Hough-space points are tested to update the silhouette. Rather than searching from the root, as for full traversal, we walk along nodes deep in the octree so as to avoid computations on interior nodes close to the root.

Our search starts with a set of octree nodes in the *active set* for frame $t$. Each node in this active set must intersect the v-sphere for frame $t$ and be an *active-set candidate*. Note that not all octree nodes are deemed to be active-set candidates. A judicious choice of the set of active-set candidates plays an important role in speeding up incremental silhouette updates; this is described in details in the next section.

For silhouette update at frame $t+1$, we perform a breadth-first search through the neighbour graph (see Figure 7), starting with nodes in the active set for frame $t$. During the search, we recurse into octree nodes partially contained in the active region, performing intersection tests between bounding volumes and the active region. For a node completely contained in the active region, all the Hough-space points therein undergo the simpler facing test against the view point. A pseudocode is given below; it takes as input the active set at frame $t$ and returns the active set at frame $t+1$, which initiates incremental search in the next frame.

```
find_incr_sil(queue* active_set) {
   queue* next_active = empty_queue();

   while (node* n = dequeue(active_set)) {
      if(checked_this_frame(n)) continue;

      if (active_region_contains(n))
         test_all_faces_in(n);
      else
         enqueue_kids(active_set, n);

      enqueue_nbrs(active_set, n);

      if (n->active_candidate &&
            intersects_current_vsphere(n))
        enqueue(next_active, n);

      mark_as_checked(n);
   }
   return next_active;
}
```

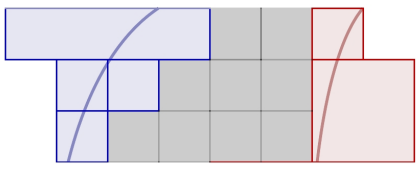


**Figure 7:** *A partial example of incremental search. The red (respectively, blue) arc is part of the boundary of the v-sphere at frame t (resp., frame $t+1$), and the red (resp., blue) nodes are from the active set for that frame. Breadth-first search proceeds from the red nodes, through the gray nodes (and their parents), and ends at the blue nodes.*

### 6.4. Selection of active-set candidates (ASCs)

If we allow incremental search to proceed along leaf (i.e., lowest-level) nodes of the octree, we may end up processing a large number of nodes during traversal of the active region. Ideally, we wish to cross the active region in few steps to avoid the overhead of many bounding box checks and queue updates. To this end, nodes at higher levels in the tree are preferred. On the other hand, if an octree node is completely contained in the active region, we need not check any of its children's bounding boxes — we can simply report all of the faces contained in its subtrees as changed. We are therefore interested in identifying nodes which are above leaves in the tree, but are still sufficiently low-level to avoid searching a large number of interior nodes. We call these nodes *active-set candidates* or *ASC*s, as we will add only these nodes to the active set at any time. We must be careful to ensure that every leaf node has an ASC above it in the octree.

A simple and reasonably effective way to choose ASCs would be to select the so-called *twig nodes* — nodes that contain a leaf child. However, twig nodes may vary in depth considerably across an unbalanced octree and may still be too small for a large mesh with a deep Hough-space octree. We thus resort to a different heuristic to produce a more suitable set of ASCs. Intuitively, we select nodes whose subtrees are well balanced, so as to take maximal advantage of hierarchical octree culling when the node intersects the active region, and whose neighbours are at the same or similar depths. We first define an ASC-cost for each node in the octree,

```
ASC-cost(node) {
   subtree_cost = node->depth / node->num_kids

   nbr_cost = 0.0
   for (n in node->neighbours)
      nbr_cost += abs(node->depth - n->depth)
   nbr_cost /= node->num_nbrs

   return w1 * subtree_cost + w2 * nbr_cost
}
```

where $w1$ and $w2$ are set by experimentation to be 2.0 and 0.25, respectively.

We identify the set of ASCs in an octree in preprocessing by first computing the ASC-cost of each node. To ensure that the candidate set stays near the leaves, we will only make a node an ASC if its children are either leaf nodes or ASCs themselves. If the ASC-cost of a node is less than the average ASC-cost of its ASC children, we remove the child nodes from the set of ASCs and insert the current node. We iterate this process until no changes to the set of ASCs are made.

## 7. Experimental results

We have tested our methods, as well as a simple dual-space silhouette extraction algorithm similar to that of Pop et al. [PDB*01], on a PC running Linux 2.6.8 with an Intel Xeon 2.80GHz processor, 2GB of memory, and an NVidia GeForce 6800 GT. Six test models, shown in Figure 8, are

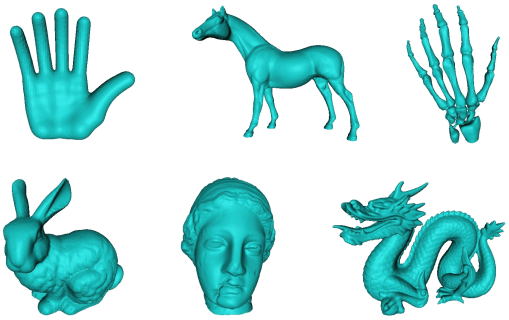


**Figure 8:** *Test models (from left to right and top-down): Hand (face count: 12K); Horse (40K); Bone (65K); Bunny (70K); Igea (268K); Dragon (300K).*

used, with face counts ranging from 12K to 300K. The primary performance measure is the number of bounding box checks (against v-spheres for intersection tests) executed, as this is the most significant elementary operation used in the silhouette extraction algorithms.

### 7.1. Static silhouette extraction

We have tested our static (initial) silhouette extraction algorithm on the test models at varying levels of details, choosing 10 random view point for each resolution and averaging the results. As we can observe from Figure 9, the number of bounding box tests in silhouette extraction scales linearly with the size of the silhouette, while the size of the model scales faster; we typically expect the silhouette size to be $O(\sqrt{n})$ relative to the model size $n$ [KW97]. While several other papers, e.g., [HZ00,SGG*00], have presented static silhouette extraction methods which experimentally scale linearly with the size of the silhouette, none of them supports incremental silhouette extraction on the same data structure.

### 7.2. Incremental silhouette extraction

We have tested our incremental silhouette extraction algorithms by moving a view point along a fixed circular path on the *xz* plane around the test models. Figure 10 shows the number of bounding box checks for three methods: full-traversal (always traversing from the root, as explained in Section 6.2) and incremental Hough-space methods and a simple full-traversal dual-space algorithm [PDB*01].

Silhouette extraction in Hough space significantly outperforms its counterpart in dual space. Incremental search through nodes near the leaves provides a noticeable and consistent improvement over the full-traversal approach, though not as great a gain as that achieved by moving from dual space to Hough space. Dual-space search has to cull away far more interior nodes than either Hough-space algorithm, and in Hough space, full-traversal search must cull more nodes

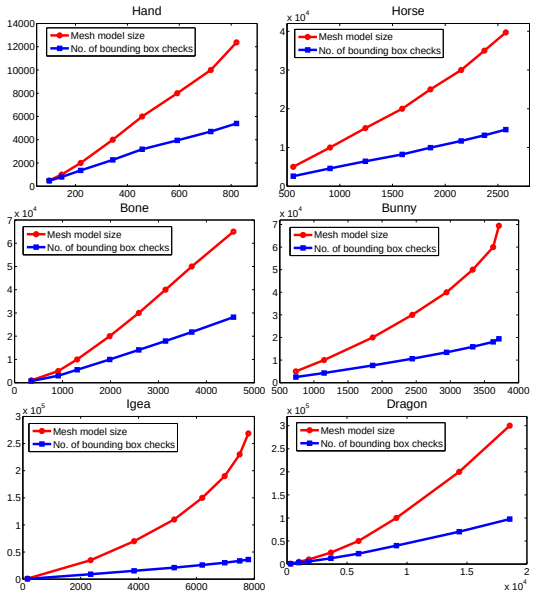



**Figure 9:** *This plot experimentally shows output sensitivity of our static silhouette extraction. Horizontal axis gives average silhouette size (computed for 10 random view points). Plotted in red is the model size and in blue the number of bounding-box checks (exhibiting a roughly linear behavior).*

than incremental update. It is also worth noting that Hough-space algorithms exhibit much more stable behavior from frame to frame, compared to the dual-space algorithm, as the octrees built around Hough-space points are more evenly balanced than those in dual space.

The three algorithms are more evenly matched in terms of the number of face checks (sidedness test against view point) performed, as shown in Figure 11 for two of the test models. Further, bounding-box checks are far more expensive than face checks, with roughly twice the number of floating-point operations and five times the number of conditional branches. Therefore, while the number of face checks performed by a silhouette extraction algorithm is important, the improved performance of our algorithms is best seen by examining bounding-box tests, as in Figure 10.

### 7.3. Histogram of leaf node depths

The performance of Hough-space algorithms relative to their dual-space counterparts may be explained by certain characteristics of the octrees generated for the respective point sets. Figure 12 shows histograms of leaf node depths for Hough-space and dual-space octrees. Specifically, we plot the number of leaf nodes at various tree depth levels. Hough-space octrees for all test models are consistently and considerably shallower, which leads to fewer interior nodes, fewer bounding box checks, and more efficient subtree culling.

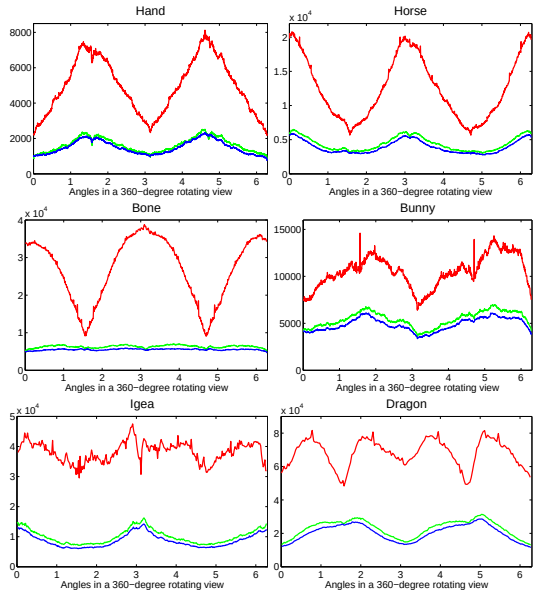



**Figure 10:** *Number of bounding box checks under incremental silhouette extraction for our test models. Results for the full-traversal Hough-space algorithm are given in green; results for incremental tree search in Hough space are given in blue; results for full-traversal in dual space are given in red. The vertical axes denote the number of checks; the horizontal axes denote the position of the view point, given by an angle in the xz plane measured from the $+x$ direction.*

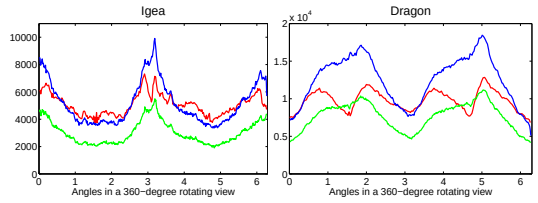



**Figure 11:** *Number of face checks under incremental silhouette extraction for two test models. Colours and axes used are the same as for Figure 10.*

## 8. Conclusion and future work

We present a complete framework for silhouette extraction in 3D Hough space. Both initial extraction of silhouettes and incremental frame-to-frame updates are efficiently implemented on a common augmented octree data structure. Our data structure is efficient to construct and enables both hierarchical traversal and low-level neighbour search; the latter is executed on a novel neighbour graph defined for octrees. Our initial silhouette extraction scheme allows for effective culling, relying on a simple characterization of mesh edges in preprocessing, and its performance is empirically shown to be linear to the size of the silhouette extracted. Switching

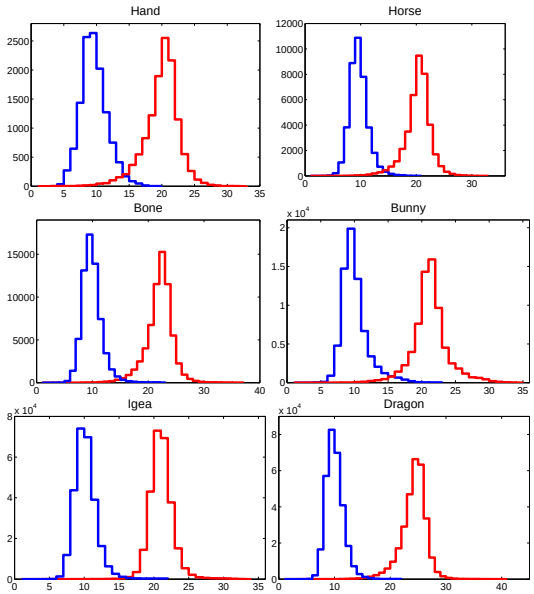



**Figure 12:** *Histogram plots for the depths of leaves in an octree. Red: dual space. Blue: Hough-space.*

from the dual space to Hough space alone results in a significant speed-up for silhouette updates. Further performance gains are consistently obtained through neighbour traversal, instead of always starting the search at the octree root. However, current improvements are seen to be marginal.

One way to improve our current approach is to optimize the search data structure with judicious choice of the origin or a set of origins; the latter appears to be a clustering problem. In the single-cluster case, an interesting problem would be to find an origin for which the resulting Hough-space points, for a given mesh, are most uniformly distributed. At the same time, it is also desirable to reduce the size of the EBVs to enable more effective culling. Both can be posed as optimization problems. Also related to the phenomenon of clustering around the origin, an alternative as our search data structure may be a spherical octree, for which denser subdivision occurs close to the origin; the resulting octree will be more balanced than our current Cartesian, axis-aligned octree. Ultimately, we wish to find a provably output-sensitive algorithm for incremental silhouette extraction under perspective projection; such an algorithm is still elusive.

Finally, none of the existing silhouette extraction algorithms lends itself to use on large, multiresolution meshes, despite of the fact that silhouette extraction on these meshes, particularly large terrains, may lead to efficient occlusion culling with much lower preprocessing cost than visibility complexes. Efficient silhouette extraction on multiresolution structures such as spline patches and subdivision surfaces

may also be of great benefit to non-photorealistic animation. We would like to adapt our methods to these structures.

**Acknowledgment:** This research is supported in part by an NSERC grant (Grant no.: 611370) from the second author. Travel funding is partially provided by the Faculty of Applied Sciences, Simon Fraser University. For mesh decimation and model display in Figure 8, the Qslim and Qvis software of M. Garland have been used.

### References

[BE99] BENICHOU F., ELBER G.: Output sensitive extraction of silhouettes from polygonal geometry. In *Proc. of Pacific Graphics* (1999), pp. 60–69.

[BKR*05] BURNS M., KLAWE J., RUSINKIEWICZ S., FINKELSTEIN A., DECARLO D.: Line drawings from volume data. *ACM Trans. Graph. 24*, 3 (2005), 512–518.

[Bla00] BLAIR D. E.: *Inversion Theory and Conformal Mapping*. American Mathematical Society, 2000.

[BS00] BUCHANAN J. W., SOUSA M. C.: The edge buffer: A data structure for easy silhouette rendering. In *Proc. of the 1st Int. Symp. on Non-Photorealistic Animation and Rendering (NPAR)* (2000), pp. 39–42.

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *ACM SIGGRAPH* (1977), pp. 242–248.

[DDSD03] DÉCORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Trans. Graph. 22*, 3 (2003), 689–696.

[DGK01] DUNCAN C. A., GOODRICH M. T., KOBOUROV S.: Balanced aspect ratio trees: Combining the advantages of *k*-d trees and octrees. *Journal of Algorithms 38* (2001), 303–333.

[GSG*99] GOOCH B., SLOAN P.-P., GOOCH A., SHIRLEY P., RIESENFELD R.: Interactive technical illustration. In *ACM Symp. on Interactive 3D Graphics* (1999), pp. 31–38.

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *In ACM SIGGRAPH* (2000), pp. 517–526.

[IFH*03] ISENBERG T., FREUDENBERG B., HALPER N., SCHLECHTWEG S., STROTHOTTE T.: A developer's guide to silhouette algorithms for polygonal models. *IEEE Comput. Graph. Appl. 23*, 4 (2003), 28–37.

[IHS02] ISENBERG T., HALPER N., STROTHOTTE T.: Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. *Computer Graphics Forum 21*, 3 (2002), 249–258.

[Jai89] JAIN A. K.: *Fundamentals of Digital Image Processing.* Prentice Hall, 1989.

[JC01] JOHNSON D. E., COHEN E.: Spatialized normal cone hierarchies. In *Proc. of the Symp. on Interactive 3D Graphics* (2001), pp. 129–134.

[KMGL96] KUMAR S., MANOCHA D., GARRETT W., LIN M.: Hierarchical back-face computation. In *Proc. of the Eurographics Workshop on Rendering Techniques* (1996), pp. 235–253.

[KW97] KETTNER L., WELZL E.: Contour edge analysis for polygonal projections. In *Geometric Modeling: Theory and Practice* (1997), Strasser W., Klein R.,, Rau R., (Eds.), pp. 379–394.

[MK04] MESETH J., KLEIN R.: Memory efficient billboard clouds for btf textured objects. In *Proc. of Vision, Modeling, and Visualization* (2004), pp. 167–174.

[MKG*97] MARKOSIAN L., KOWALSKI M. A., GOLDSTEIN D., TRYCHIN S. J., HUGHES J. F., BOURDEV L. D.: Real-time nonphotorealistic rendering. In *ACM SIGGRAPH* (1997), pp. 415–420.

[PDB*01] POP M., DUNCAN C., BAREQUET G., GOODRICH M., HUANG W., KUMAR S.: Efficient perspective-accurate silhouette computation and applications. In *Proc. of Annual Symp. on Computational geometry* (2001), pp. 60–68.

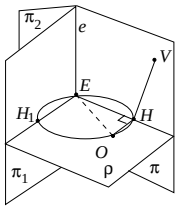
[PF02] PLAENKERS R., FUA P.: Model-based silhouette extraction for accurate people tracking. In *European Conf. on Computer Vision* (2002), pp. 325–339.

[RC99] RASKAR R., COHEN M.: Image precision silhouette edges. In *Proc. of the Symp. on Interactive 3D Graphics* (1999), pp. 135–140.

[SGG*00] SANDER P. V., GU X., GORTLER S. J., HOPPE H., SNYDER J.: Silhouette clipping. In *ACM SIGGRAPH* (2000), pp. 327–334.

[ZP01] ZAHARIA T., PRETEUX F.: Hough transform-based 3d mesh retrieval. In *Proc. of the SPIE Conf. 4476 on Vision Geometry X* (2001), pp. 175–185.

## Appendix




**[Proof of Theorem 2:]** Refer to figure to the right. First assume that $e$ is a silhouette edge with respect to view point $V$. Then as the support plane $\pi_2$ rotates about the edge $e$ towards $\pi_1$, $V$ will be hit. Let $\pi$ be the "hit" plane. Let $H$ be the Hough transform of $\pi$, which lies along the Hough transform of $e$. Let $O$ be the origin. Then the Hough circle of $e$ passes through $O$ and $H$. It follows that $OH \perp EH$. As the support plane $\rho$ of the Hough circle is penpendicular to $\pi$, $OH \perp \pi$. Thus $OH \perp HV$ and $H$ must lie on the v-sphere determined by $V$. Conversely, if the v-sphere intersects the Hough transform of $e$ at $H$, then $OH \perp EH$ and $OH \perp HV$. It follows that $OH \perp \pi$ and thus $\rho \perp \pi$. Hence the plane $\pi$ passes through $e$. Since $H$ is on the Hough transform of $e$, $\pi$ is an intermediate rotating plane that hits $V$, which implies that $e$ is a silhouette edge with respect to $V$. Finally, it is not hard to see that the tangency case in the theorem occurs when $\pi$ passes through $O$.