On Computing Exact Visual Hulls of Solids Bounded by Smooth Surfaces

Svetlana Lazebnik
Beckman Institute
University of Illinois, Urbana, USA
slazebni@uiuc.edu

Edmond Boyer
Movi–Gravir–Inria Rhône-Alpes
Montbonnot, France
Edmond.Boyer@inrialpes.fr

Jean Ponce
Beckman Institute
University of Illinois, Urbana, USA
ponce@cs.uiuc.edu

Abstract

This paper presents a method for computing the visual hull that is based on two novel representations: the rim mesh, which describes the connectivity of contour generators on the object surface; and the visual hull mesh, which describes the exact structure of the surface of the solid formed by intersecting a finite number of visual cones. We describe the topological features of these meshes and show how they can be identified in the image using epipolar constraints. These constraints are used to derive an image-based practical reconstruction algorithm that works with weakly calibrated cameras. Experiments on synthetic and real data validate the proposed approach.

1. Introduction

Most algorithms for surface reconstruction from outlines compute some form of the *visual hull* [10], or the intersection of solid visual cones formed by back-projecting silhouettes found in the input images. The basic approach dates back to Baumgart's 1974 PhD thesis [1], where a polyhedral visual hull is constructed by intersecting the viewing cones associated with polygonal silhouettes. Volume intersection has remained the dominant paradigm for decades, implemented using representations as diverse as octrees [15] and triangular splines [14]. More recently, graphics researchers have presented efficient algorithms that avoid general 3D intersections by taking advantage of epipolar geometry [11, 13]. Given an image sequence from a camera undergoing a continuous motion, it is also possible to avoid explicit intersections by reconstructing the visual hull as the envelope of the surface tangent planes along the smoothly deforming occluding contours [2, 3, 16].

Defined in full generality, the visual hull is the maximal shape consistent with an object's silhouettes as seen from any viewpoint in a given region, and the *exact* visual hull is the visual hull with respect to a continuous region of space surrounding the object [10]. In this paper we do not treat such limiting cases, but consider the visual hull associated with a finite number of isolated viewpoints (in this discrete formulation, the volume intersection and the silhouette-consistency definitions are equivalent). We represent the visual hull as a generalized polyhedron: the faces on its surface are visual cone patches, edges are intersection curves between two viewing cones, and vertices are isolated points where more than two faces meet. In this context, a visual hull is exact when it correctly captures the connectivity of these features. Based on this notion of exact visual hulls, we specify a novel reconstruction algorithm that does not rely on polyhedral intersections or voxel-based carving, and produces precise topological and geometric meshes.

2. Preliminaries

We assume that we are observing a solid object with weakly calibrated pinhole cameras. The surface of the object is smooth and without planar patches, and the cameras are in general position. It is also assumed that apparent contours of the object have been identified in each input view and oriented counterclockwise, so that the image of the object always lies to the left of the contour. To simplify the presentation, we also assume that contours do not contain singularities such as T-junctions and cusps, and restrict our attention to objects of genus 0.

In the rest of the paper, we use the following terminology. The *rim* or *contour generator* associated with a camera is the set of all surface points where the optical ray through the pinhole grazes the object. For general viewpoints, the rim is a smooth space curve without singularities [4]. Two rims can intersect at isolated points on the surface, called *frontier points* [3, 8, 12], where the viewing rays from both pinholes lie in the surface tangent plane. The projection of a rim onto the image plane of a camera is the *apparent contour*. The set of rays from one camera center passing through points on the surface forms the *visual cone* associated with that camera. As described in the introduction, the solid formed by the intersection of all given viewing cones is the *visual hull*. Note that the shape of the viewing cones depends only on the camera center and on the shape of the object, not on the position of the image plane. Thus, projective geometry is sufficient to describe the structure of the

visual hull. In particular, it is possible to develop a visual hull algorithm that relies only on weak calibration.

Imagine sweeping out a cone of optical rays along one apparent contour. Since each ray must graze the object along the rim, each ray must lie on the visual hull for some non-empty interval around its point of tangency with the object surface. In this way, each ray contributes an interval to the surface of the visual hull, and the collection of these intervals along all rays forms a *cone strip* that continuously bounds the rim on each side. Strips are delimited by segments of *intersection curves* between pairs of visual cones. An intersection curve generally does not lie on the surface, except at frontier points, where the tangent planes to the two cones and to the object coincide [5]. At these points, the intersection curve is singular: it has four branches that converge to create a characteristic X-shape (see Figure 1).

Figure 2 shows an example of an ovoid observed by three cameras. Three viewing cones are drawn, along with intersection curves and rims. The figure shows frontier points and *triple points* where three viewing cones intersect. In the figure, each frontier point is incident to four rim arcs and four intersection curve branches, and each triple point is incident to six intersection curve branches, only three of which belong to the visual hull. It can be shown that these incidence relations hold in general.

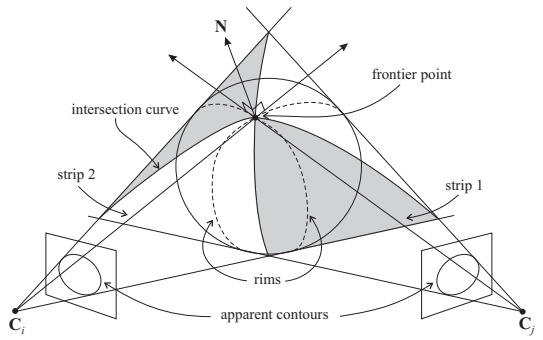

Figure 1: A surface observed by two cameras. The two rims intersect at the frontier point where two cone strips cross.

Now we introduce the two meshes computed by our algorithm. The *rim mesh* is defined on the surface of the actual object. Its vertices are frontier points, edges are rim segments between two successive frontier points, and faces are regions of the surface bounded by two or more edges. A conceptual precursor of the rim mesh is the *epipolar net* of Cross and Zisserman [5], who informally discuss, but do not construct, the arrangement of rims on the surface of an object as the camera moves. The *visual hull mesh* is a topological description of the configuration of visual cone patches on the surface of the solid formed by the intersection of all given visual cones. Its vertices are frontier points (where two strips cross) and triple points (where three cones intersect), edges are intersection curve segments between two consecutive vertices, and faces are the cone patches that make up the strips. Examples from Figures 1 and 2 illustrate the fact that successive frontier points on one rim break up the cone strip along that rim into separate faces. Thus, there exists a one-to-one relationship between edges of the rim mesh and faces of the visual hull mesh. Continuing with the example of Figure 2, Figure 3 shows the rim and visual hull meshes of the ovoid.

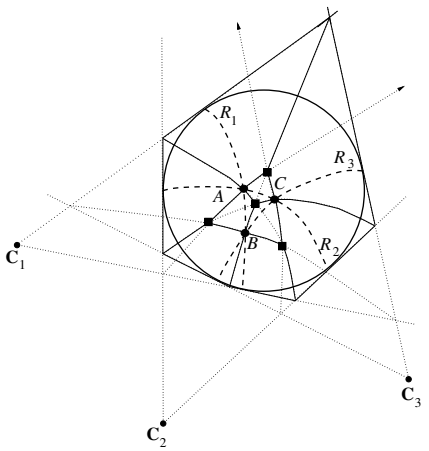

Figure 2: Configuration of rims and intersection curves for an ovoid observed by three cameras. Rims are dashed arcs, frontier points are labeled dots, and triple points are squares. Dotted arcs are intersection curve branches outside the visual hull.

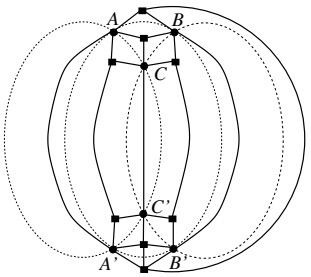

Figure 3: The rim and visual hull meshes of the ovoid from Figure 2. Frontier points are circles, and triple points are squares. Rim segments (edges of the rim mesh) are dashed. Intersection curve segments (edges of the visual hull mesh), are bold lines. Note that frontier points A', B', and C' belong to the region of the surface not visible in the previous figure.

3. Computing the Rim Mesh

We begin by computing the frontier points, which are the vertices of the rim mesh. At a frontier point $\mathbf{P}_{ij}$ due to views i and j, the tangent plane to the surface is also the epipolar plane determined by $\mathbf{P}_{ij}$ and the camera centers $\mathbf{C}_i$ and $\mathbf{C}_j$. In the images, this means that corresponding epipolar lines $\mathbf{l}_{ij}$ and $\mathbf{l}_{ji}$ are both tangent to the respective contours at the projections $\mathbf{p}_i$ and $\mathbf{p}_j$ of $\mathbf{P}_{ij}$. Figure 4 illustrates this basic setup, along with other notation that we will need later. Finding a pair of matching frontier points in images i and j is a one-dimensional search problem, where

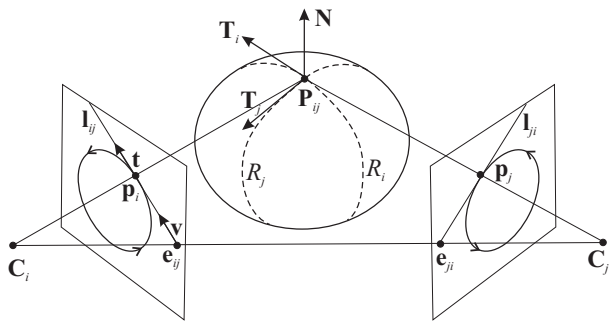

Figure 4: $\mathbf{P}_{ij}$ is a frontier point where two rims R_i and R_j intersect. $\mathbf{P}_{ij}$ projects onto points $\mathbf{p}_i$ and $\mathbf{p}_j$ at which the respective contours are tangent to the two matching epipolar lines $\mathbf{l}_{ij}$ and $\mathbf{l}_{ji}$.

we parametrize the pencil of epipolar lines by their slope and look for line $\mathbf{l}_{ij}$ that is tangent to the ith contour, such that the corresponding line $\mathbf{l}_{ji}$ in the jth image is tangent to the jth contour. In the presence of contour extraction and calibration errors, there may not exist a pair of matching epipolar lines that exactly satisfy the tangency constraint. In this situation, we find approximately matching frontier points such that the angle difference between the tangent line in one image and the reprojected epipolar tangent from the other image is minimized. Difficulties caused by data error will be further discussed in Section 4.

We obtain all frontier points by matching their projections in two images. The four rim edges incident to a particular frontier point are given by intervals on the two apparent contours that are incident to this point. Thus, there exists a one-to-one correspondence between contour segments in the images and edges of the rim mesh. The orientation of rim edges is then given by the orientation of the corresponding contour segments. In this way, we obtain the complete adjacency information for edges and vertices of the rim mesh. To compute the faces, we need to know the relative ordering in space of the four rim segments incident on each vertex. More formally, we associate with each frontier point $\mathbf{P}_{ij}$ a circular list I where the four edges appear in CCW order around the surface normal. Let $\mathbf{T}_i$, $\mathbf{T}_j$ be the tangents to the rims R_i and R_j at $\mathbf{P}_{ij}$ (see Figure 4). Then the index i appears before the index j in the ordered list I iff

$$(\mathbf{T}_i \wedge \mathbf{T}_j) \cdot \mathbf{N} > 0, \tag{1}$$

where $\mathbf{N}$ is the outward-pointing surface normal (computed as the cross product of the oriented tangent to the contour and the viewing direction).

Equation (1) gives rim ordering in terms of of tangents to the apparent rims, which cannot be computed given only image information. Therefore, we need an equivalent image-based expression for rim ordering. Consider image i and let $\mathbf{v}$ be the direction from the epipole $\mathbf{e}_{ij}$ to $\mathbf{p}_i$, the projection of $\mathbf{P}_{ij}$. Let also $\mathbf{t}$ be the tangent to the contour at $\mathbf{p}_i$ (see Figure 4), and let k_s be the apparent curvature at $\mathbf{p}_i$. Then rims R_i and R_j are CCW oriented at $\mathbf{P}_{ij}$ iff

$$(\mathbf{v} \cdot \mathbf{t})\ k_s > 0. \tag{2}$$

The above expression is equivalent to (1), and the following is a brief sketch of the proof. When the viewing direction rotates in the surface tangent plane around the normal at $\mathbf{P}_{ij}$, the tangent to the rim also rotates. The directions of camera and tangent rotation are the same if the surface is elliptic at $\mathbf{P}_{ij}$ and opposite if the surface is hyperbolic. This is a direct consequence of the fact that the Gauss map of the surface is orientation-preserving at elliptic points and orientation-reversing at hyperbolic points [6]. Thus two rims R_i and R_j are CCW oriented if (a) the camera rotates CCW around the normal from position i to j and the surface is elliptic at $\mathbf{P}_{ij}$; or (b) the camera rotates CW and the surface is hyperbolic. The sign of $\mathbf{v} \cdot \mathbf{t}$ is positive if the camera rotates CCW in the tangent plane and negative otherwise. Moreover, the sign of the apparent curvature k_s is positive if the surface is elliptic or negative if the surface is hyperbolic [9]. Thus, expression (2) is positive in the two above mentioned cases and negative otherwise. It is therefore the desired image-based expression equivalent to (1).

Given the above rim ordering criterion, it is straightforward to trace the loops of edges bounding rim faces. Suppose we start with one rim segment s of the rim R_i and want to find the face that lies to its left. Informally, we traverse s along its direction and at its endpoint $\mathbf{P}_{ij}$, simply take a left turn to get to the next edge, that is, we select the edge preceding s in the ordered circular list of $\mathbf{P}_{ij}$. We move from endpoint to endpoint in this manner, traversing edges either forward or backward along their orientation, taking a left turn each time until we complete a cycle.

4. Computing the Visual Hull Mesh

As demonstrated in the previous section, we can compute the topology of the rim mesh without knowing anything about its geometry, except for the positions of frontier points (note that under weak calibration we can only recover these positions up to a projective transformation). This topology constrains the adjacency relationships between triple points and intersection curves, which are the vertices and edges of the visual hull mesh.

A triple point $\mathbf{P}_{ijk}$ the intersection of three optical rays back-projected from contour points $\mathbf{p}_i$, $\mathbf{p}_j$, and $\mathbf{p}_k$ in three different images (see Figure 5). In particular, $\mathbf{p}_k$ satisfies the *transfer equation* [7]

$$\mathbf{p}_k = \mathbf{l}_{ki} \wedge \mathbf{l}_{kj} = (\mathbf{F}_{ik}\, \mathbf{p}_i) \wedge (\mathbf{F}_{jk}\, \mathbf{p}_j), \tag{3}$$

where $\mathbf{F}_{mn}$ is the *fundamental matrix* mapping points in image m to epipolar lines in image n, and homogeneous

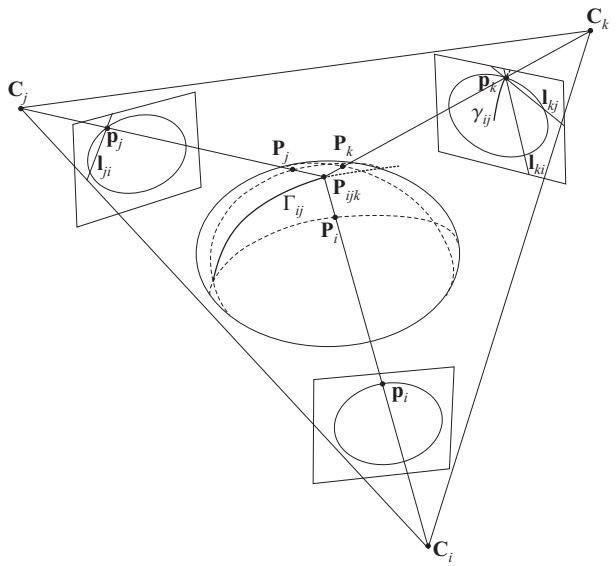

Figure 5: The triple point $\mathbf{P}_{ijk}$ is a "phantom point" where the rays formed by back-projecting epipolar correspondents $\mathbf{p}_i$, $\mathbf{p}_j$, and $\mathbf{p}_k$ meet in space. $\mathbf{P}_{ijk}$ can be located by tracing the intersection curve Γ_{ij} between views i and j and noticing when its projection γ_{ij} in the kth image crosses the contour.

coordinates are used for image points. Points $\mathbf{p}_i$ and $\mathbf{p}_j$ satisfy symmetric equations. Any pair of $\mathbf{p}_i$, $\mathbf{p}_j$ and $\mathbf{p}_k$ are *epipolar correspondents* — that is, any two of the points lie in the epipolar plane defined by the two camera centers and one of the points [2]. A triple point is a standard trinocular stereo correspondence, but it does not usually lie on the surface because $\mathbf{p}_i$, $\mathbf{p}_j$ and $\mathbf{p}_k$ are projections of three different points $\mathbf{P}_i$, $\mathbf{P}_j$, and $\mathbf{P}_k$ on three different rims.

Just as with finding frontier points, finding triple points is a one-parameter search. We walk along the ith contour in discrete steps and for each contour point $\mathbf{p}_i$ find the epipolar line $\mathbf{l}_{ji} = \mathbf{F}_{ij}\,\mathbf{p}_i$ in image j, and locate an epipolar correspondent $\mathbf{p}_j$ by intersecting $\mathbf{l}_{ji}$ with the jth contour. We then obtain a third point $\mathbf{p}_k$ by transferring $\mathbf{p}_i$ and $\mathbf{p}_j$ using (3) and check whether $\mathbf{p}_k$ lies on the kth contour. As shown in Figure 5, transfer of successive epipolar correspondents allows us to trace the intersection curve Γ_{ij} between ith and jth cones in the kth image. The triple point is revealed when the traced curve crosses the kth contour.

In general, epipolar correspondents are not unique. In the case shown in Figure 5, each epipolar line intersects each contour twice (this reflects the fact that intersection curves have multiple branches). Moreover, the epipolar correspondence criterion does not say when a triple point belongs to the visual hull — the additional constraint is that the point must not project outside the silhouette in any other input view. However, if we are able to exactly compute the positions of frontier points along the contours, we can use this information to simplify the search for triple points.

Each face of the rim mesh is bounded by rim segments that also belong to the surface of the visual hull, so we can identify faces with regions on the surface of the visual hull that have the same boundary. Consider a single face f of the rim mesh. The edges of f tell us which viewing cones contribute to the visual hull inside the region identified with f, and give us a corresponding subset of contours that need to be searched for triple points belonging to this region. Since each edge of f corresponds to a single contour interval in some image, any triple point that belongs to f must project to a point along these intervals. Thus, it is sufficient to trace intersection curve segments between each pair of these intervals and find triple points when the curve being traced goes outside the contour in one of the other views that contribute to f. Taking each face separately, we compute triple points, intersection curves, and their connectivity. Since we know which pair of cones gave rise to each segment of an intersection curve, we can identify all the segments bounding a cone strip. Individual faces of the strips are identified by grouping all the intersection curve segments that project within the contour interval that corresponds to a particular rim segment.

The algorithm described above yields the correct visual hull mesh given exact input data (perfectly extracted contours, error-free fundamental matrices). However, noise and calibration error tend to destroy exact topological features, most importantly, intersection curve crossings at frontier points. Figure 6 illustrates the situation. The epipolar line $\mathbf{l}_{ij}$ is tangent to the contour at point $\mathbf{p}_i$ in the ith image. This line corresponds to the line $\mathbf{l}_{ji}$ in the jth image, which is not tangent to the jth contour, but intersects it in two epipolar correspondents $\mathbf{p}_{j1}$ and $\mathbf{p}_{j2}$. Intersecting the visual rays due to these three points in the epipolar plane yields two distinct intersection curve points $\mathbf{P}_{ij1}$ and $\mathbf{P}_{ij2}$, instead of a single frontier point. Thus, instead of being singular, the intersection curve separates into two distinct branches that do not meet. In order to approximate the position of a frontier point, we have to match $\mathbf{p}_i$ with the epipolar tangency point $\mathbf{p}_j$ in the jth image. However, the two points do not lie in the same epipolar plane, and visual rays through them do not intersect. We could estimate the location of $\mathbf{P}_{ij}$ as the midpoint of the segment connecting the points of closest approach of the two rays, but this approximated point does not lie on the traced intersection curves. This leads to serious consistency problems for a naive implementation that attempts to strictly enforce combinatorial constraints on exact visual hull structure.

Intuitively, small perturbations to exact contour and calibration data result in contours that back-project to general cones in space, the intersection of which does not have to share the properties of exact visual hulls. For instance, while we know that cone strips never break up in theory (each ray interval along the strip must contain at least one point), in noisy data, they may break up near the frontier points as shown in Figure 6. Nevertheless, even with large

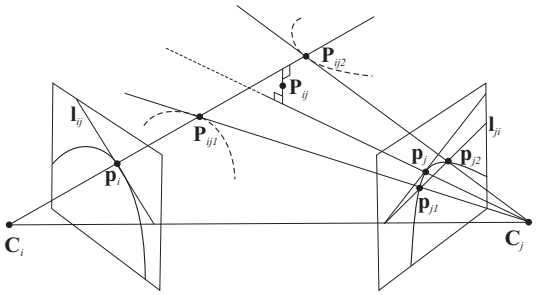

Figure 6: Tracing intersection curves given inexact data (see text). Intersection curve branches are shown as dashed lines.

errors in the data, the intersection of cones in space is still well defined, and we can compute it using a variant of our exact algorithm. We trace entire intersection curves, as opposed to breaking them up into pieces belonging to separate faces of the rim mesh, and then clip out all components of the curves that project outside any of the silhouettes. In the process, combinatorial information about the curves is maintained, so that it becomes possible to recover the geometry of cone strips. Namely, boundary points of the strips are connected in the order induced by the contour parametrization, and are separated into two groups that correspond to segments bounding the rim on the near and the far side with respect to their distance from the camera. This data structure is a monotone polygon, and it can be triangulated in linear time for purposes of display.

5. Experimental Results

The first input sequence consists of six synthetically generated images of an egg model. Contours were extracted using snakes and modeled as cubic B-splines. As seen in Figure 7, the algorithm correctly generates the rim mesh and the visual hull, both in their topological and geometric form. The contrast between these two forms is clearly visible by comparing two renderings of the same strip in Figure 7 (f) and (g). The exact strip explicitly shows frontier points and triple points, and intersection curves are forced to converge in four branches at frontier points. The triangulated strip does not degenerate at frontier points, because the robust strip tracing algorithm ignores them.

We also demonstrate results for two calibrated nine-image turntable sequences of a gourd and a vase (Figure 8). For both of these data sets, the algorithm constructs a complete rim mesh, even though many of the frontier points are densely clustered near the top and the bottom. To better visualize the structure of these meshes, we rendered their vertices and edges as graphs using a publicly available graph drawing program. The graphs, shown in Figure 8 (b) and (h), reveal the regular structure of rim crossings which is impossible to observe in the images themselves. Each rim mesh in our examples obeys Euler's formula for topological polyhedra of genus 0, $V + F = E + 2$ (note that the meshes are actually planar, even though the graph layouts shown are not). Because of stability problems inherent in real-world data, the algorithm does not recover topological visual hull meshes for these two data sets. Instead, we obtain precise geometric models of the visual hulls that do not capture every triple point, but are suitable as input for common modeling and rendering applications. Figure 8 shows these models, along with selected strips. Note that the strips degenerate completely for relatively large intervals near the top and the bottom, in the areas of dense frontier points. This behavior is not possible in theory, but it occurs in practice, as discussed in Section 4.

6. Discussion and Future Work

Our preliminary results are intriguing. Significantly, the recovery of exact rim meshes has proven to be robust even with densely clustered frontier points that do not lie on matching epipolar lines. Note that the rim mesh structure depends only on the relative ordering of frontier points along the rims, not on absolute positions — hence the relative stability of the topology. With visual hull meshes, the situation is different: the connectivity of intersection curves and triple points is elusive, while the geometry may still be recovered reliably. It will be important to investigate the question of whether these instabilities are inherent in the conditioning of exact visual hull computation, or are introduced by our algorithm. We are considering a different approach to recovering the topology of the visual hull mesh that would take full advantage of the combinatorial constraints given by the structure of the rim mesh. We are also working on extending our implementation to deal with T-junctions and surfaces of arbitrary genus, to handle more complex and visually interesting objects.

Overall, our approach has several attractive features. Most importantly, it is based on an analysis of the exact structure of visual hulls from finitely many viewpoints, which has received little attention in previous research. Our approach takes advantage of epipolar geometry and weak calibration — in a sense, we don't need to know where the cameras are. Moreover, our algorithm uses only two-dimensional computations, and constructs a range of shape representations, from graph-theoretic and topological, to completely image-based, to purely geometric. For these reasons, it brings fresh insights to the theory and practice of the venerable problem of visual hull computation.

Acknowledgments. This work was supported in part by the Beckman Institute, the National Science Foundation under grant IRI-990709 and a UIUC-CNRS collaboration agreement.

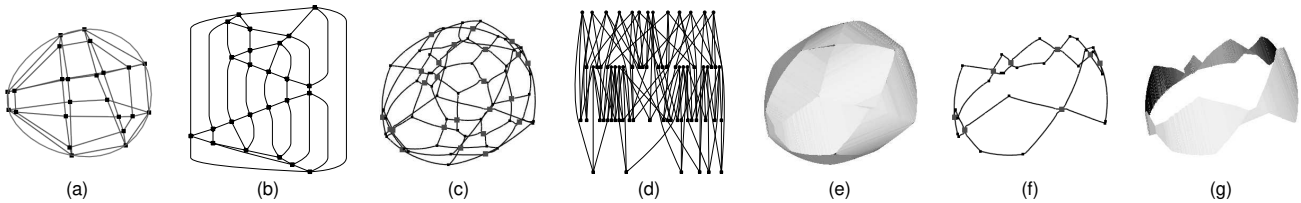

Figure 7: Egg results. (a) the rim mesh superimposed on one of the input outlines; (b) the vertices and edges of the rim mesh shown as a graph (24 frontier points, 48 edges, 26 faces); (c) the visual hull mesh from shown from a viewpoint not in the input set; (d) a graph of the visual hull vertices and edges (68 vertices, 114 edges, 48 faces); (e) a triangulated geometric model of the mesh; (f) one of the strips making up the exact visual hull mesh; (g) a triangle model of the same strip.

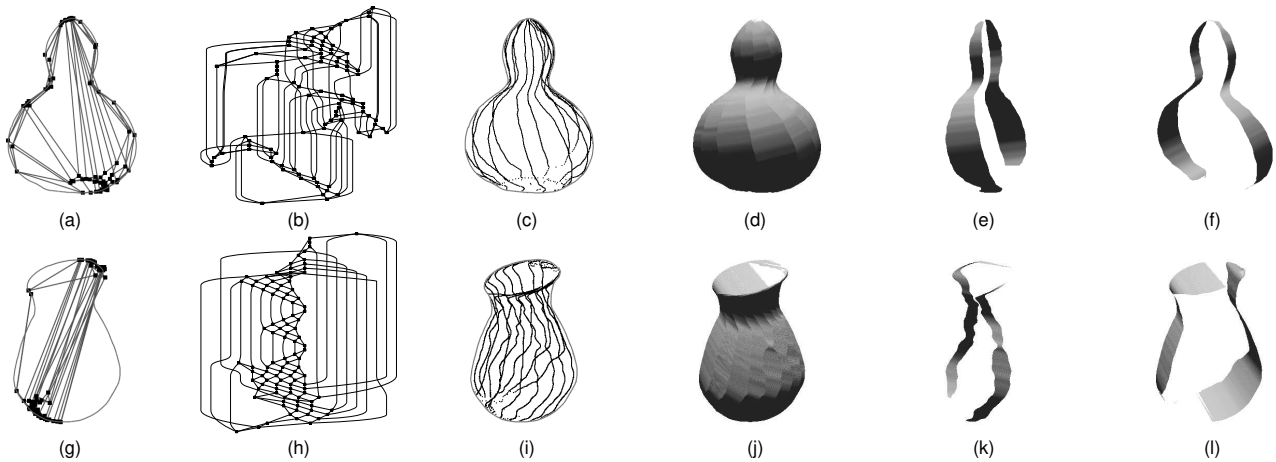

Figure 8: Top row: gourd results. (a), (b) the rim mesh and corresponding graph (96 vertices, 192 edges, 98 faces); (c) intersection curves that make up the visual hull; (d) triangulated visual hull model; (e), (f) two of the strips from the model. Bottom row: teapot results. (g), (h) rim mesh (104 vertices, 208 edges, 106 faces); (i) intersection curves on the surface of the visual hull; (j) visual hull model; (k), (l) two cone strips.

References

[1] B.G. Baumgart, "Geometric Modeling for Computer Vision", Ph. D. Thesis (Tech. Report AIM-249), Stanford University, 1974.

[2] E. Boyer and M. Berger, "3D Surface Reconstruction Using Occluding Contours", *Int. J. Comp. Vision*, 22(3), 1997, pp. 219-233.

[3] R. Cipolla, K. Astrom, and P.J. Giblin, "Motion from the Frontier of Curved Surfaces", *Proc. IEEE Int. Conf. on Comp. Vision*, 1995, pp. 269-275.

[4] R. Cipolla and P.J. Giblin, *Visual Motion of Curves and Surfaces*, Cambridge University Press, Cambridge, 1999.

[5] G. Cross and A. Zisserman, "Surface Reconstruction from Multiple Views Using Apparent Contours and Surface Texture", *NATO Advanced Research Workshop on Confluence of Computer Vision and Computer Graphics*, 2000, pp. 25-47.

[6] M. do Carmo, *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[7] O. Faugeras and L. Robert, "What Can Two Images Tell Us About a Third One?", *Int. J. Comp. Vision*, 18(1), 1996, pp. 5-19.

[8] P.J. Giblin and R. Weiss, "Epipolar Curves on Surfaces", *Image and Vision Computing*, 13(1): pp. 33-44, 1995.

[9] J. Koenderink, "What Does the Occluding Contour Tell Us About Solid Shape?", *Perception*, 1984, pp. 321-330.

[10] A. Laurentini, "The Visual Hull Concept for Silhouette-based Image Understanding", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(2), 1994, pp. 150-162.

[11] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan, "Image-based Visual Hulls", *Proc. SIGGRAPH* 2000, pp. 369-374.

[12] J. Porrill and S. Pollard, "Curve Matching and Stereo Calibration", *Image and Vision Computing*, 9(1): pp. 45-50, 1991.

[13] I. Shlyakhter, M. Rozenoer, J. Dorsey, and S. Teller, "Reconstructing 3D Tree Models from Intrumented Photographs", *IEEE Computer Graphics and Applications*, 21(3), 2001, pp. 53-61.

[14] S. Sullivan and J. Ponce, "Automatic Model Construction, Pose Estimation, and Object Recognition from Photographs using Triangular Splines", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(10), 1998, pp. 1091-1096.

[15] R. Szeliski, "Rapid Octree Construction From Image Sequences," *CVGIP: Image Understanding*, 1(58), 1993, pp. 23-32.

[16] R. Vaillant and O. Faugeras, "Using Extremal Boundaries for 3-D Object Modeling", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(2), 1992, pp. 157-173.