Visual Hull Graphs
Matching and Compressing Sequences of Visual Hulls

by

Naveen Goela

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by . . . .
Trevor Darrell
Associate Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Visual Hull Graphs

Matching and Compressing Sequences of Visual Hulls

by

Naveen Goela

Abstract

In this thesis, we implement the polyhedral visual hull (PVH) algorithm in a modular software system to reconstruct 3D meshes from 2D images and camera poses. We also introduce the new idea of *visual hull graphs*. For data, using an eight camera synchronous system after multi-camera calibration, we collect video sequences to study the pose and motion of people. We generate visual hulls (VHs) from these video sequences to capture 3D shape. To efficiently process VH meshes, we use wavelet compression of 2D input contours and show how smooth subdivision surfaces may approximate irregular meshes. In order to define a visual hull motion graph, we use a simple distance metric for 3D pose by calculating Chamfer distances between 2D shape contours. At each frame of our graph, we store a view independent 3D pose and calculate the transition probability to any other frame based on similarity of pose. New realistic motion is synthesized by walking through cycles in the graph. Our results are new videos of arbitrary length and viewing direction based on single sample source video.

Thesis Supervisor: Trevor Darrell
Title: Associate Professor

Acknowledgments

To all my M.Eng friends at MIT. We finally graduate.

To Professor Darrell for his advice and vision.

To Greg and Kristen for thesis related guidance on visual hulls and contour matching.

To Mario for somehow compressing images from 8 cameras in our camera setup, and for answering all my questions.

To Mike for his help with cameras and images, and also for answering all my questions.

To Ali and Louis-Phillipe and Neal and Kevin and David for help in lab.

To Patrick, Ben, Brian, and Ellie for company during late nights both in the old and new buildings.

To the new Stata Center and the people of CSAIL for years of research to come.

To my mom, dad, and brother for unconditional support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, vision research has moved quickly with the digital age. Interfaces between people and technology are advanced and intelligent. Applications include 3D tele-conferencing, speech and gesture recognizing agents, and 3D laser guided surgery. As devices become more complex with access to information databases, intuitive human-computer interaction is essential in bridging the gap.

In visualizing and grasping information, humans often do best in the 3D domain. Interaction in the 3D domain is our normal experience with the environment. If our communication with computers or robotic agents in the future were through tangible realistic interfaces, it would not be surprising. For this reason, much research energy is along channels of artificial intelligence, human-computer interaction, and robotics research.

While a picture is worth a thousand words, a model or replica of the objects in the picture is even better. Medical scans of tissue reveal details over cross sections but a 3D model offers more precision and information. Watson and Crick won the Nobel Prize after visualizing 3D helical bands of DNA from 2D X-ray crystallographic blueprints.

In other areas such as long distance communication, 3D interaction is now possible. Telephone communication has been the norm. However, more interactive experiences through web video streams are sold commercially today. 3D teleconferences will soon be a form of standard communication. Research for reconstructing geometry and texture for teleconferencing is an active area in academia and industry [3] [1].

In this thesis, we present a method for acquiring 3D shape from 2D images and camera poses. The polyhedral visual hull (PVH) algorithm described in **Chapter 2** extracts shape

from silhouettes in the form of 3D meshes. We develop our own implementation of the algorithm based on work by Matusik, et. al. [14] [16]. Our system is useful for building sequences of hulls from input video.

The idea of visual hulls is immediately applicable for low-cost object digitization and CAD modelling. In addition, the visual hull is popular in graphics applications as a view independent representation. We may construct realistic avatars and place them in various dynamic scenes. We may watch a football play from a new synthesized viewing angle. Recent movies show panoramic video scenes using multi-view data from many cameras.

As 3D information is important, the next generation of complex scanners and modelling tools are being built. The increasing complexity of graphics meshes has created the need for compression algorithms and multi-resolution analysis. We would like to view a mesh from coarse to fine representations for rendering efficiency. One application is the multi-resolution of ground terrain for aircraft. In **Chapter 3**, we experiment with ways to wavelet-encode silhouette contours for visual hull compression. We also produce subdivision surfaces to smoothly approximate the irregular 3D hull.

Once we have developed a method to acquire 3D shape, what further analysis is possible and relevant? From a vision standpoint, if we know the geometric shape of an object, we can use it for object recognition or motion tracking. For example, with multi-view data of a person, we can learn walking patterns and build classifiers based on gait. For face recognition, if we have a 3D model of the face, we can build a more accurate recognizer by accounting for illumination and pose variation.

In this thesis, we analyze patterns of pose and motion by constructing a visual hull (motion) graph from input video. In **Chapter 4**, we show how to capture video sequences after multi-camera synchronization and calibration. Having constructed sequences of visual hulls, in **Chapter 5** we define similarity of pose through 2D silhouettes and use a Chamfer distance metric to obtain an adjacency representation for the visual hull graph. To test the graph, we synthesize new videos of arbitrary length based on cycles.

In the upcoming chapters, each subproblem could be expanded into an area of research. Automatic camera calibration, epipolar and multi-view geometry, visual hull acquisition, 3D modelling, mesh compression, rendering, shape contour matching, skeleton feature extraction, pose inference and motion graphs are the focus of recent research. We provide our own insight into these topics as encountered through the engineering of our system.

Chapter 2

Polyhedral Visual Hull

In this chapter, we describe an implementation of the polyhedral visual hull (PVH) algorithm. Visual hull *shape from silhouette* construction is a popular method of shape estimation. The algorithm constructs an upper bound on the shape of an object based on multiple source views. The idea is used in applications such as non-invasive 3D object digitization, 3D object recognition and more recently - face recognition, human motion tracking and analysis.

To gain intuition, imagine a shadow play as shown in Figure 2-1. The audience infers shape and pose from the motion of the character's silhouette. The shadow of the character yields information about the actual shape. In a more constrained problem, if we have a rigid puppet and turn the puppet at known angles while watching its shadow, should we be able to approximate its 3D shape?

Figure 2-1: A silhouette from a shadow play.

The visual hull problem is a generalization of this scenario. Instead of knowing the angles of a rigid puppet as it turns, we use K simultaneous views of the puppet as the *input* to the PVH algorithm. Each view consists of an image taken from a camera at a particular

time t_i. The *output* of the algorithm is a triangular mesh that represents the 3D shape of the scene at time t_i based on contours found in the input images. In this case, the object(s) in the scene need not be rigid objects because at each time t_i we have multiple synchronized views for reconstruction.

The following sections describe the reconstruction process as implemented in a software system for building visual hulls. Section 2.1 contains an overview with visual hull definition, previous methods, and a comparison of sampling vs. polyhedral construction. In Sections 2.2 and 2.3 we describe the camera model as well as the silhouette approximations through contours. The PVH algorithm is outlined from Section 2.4 to Section 2.9. The conclusion addresses disadvantages of the visual hull approach and offers new ideas for applications.

2.1 Overview

2.1.1 Visual Hull

The concept of the visual hull was first proposed in 1994 by Laurentini [12]. The visual hull is the maximal silhouette-consistent shape with respect to a given viewing region. As shown in Figure 2-2, the boundary rays projected from each silhouette constrain the shape of the visual hull from different views.
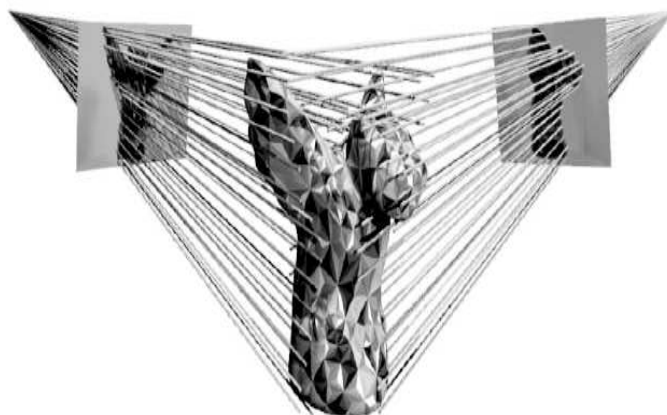
Figure 2-2: The visual hull is the maximal silhouette-consistent shape.

From a computational geometry perspective, the visual hull is a volume formed from the intersection of silhouette cones or extruded silhouettes. Suppose a scene is viewed from
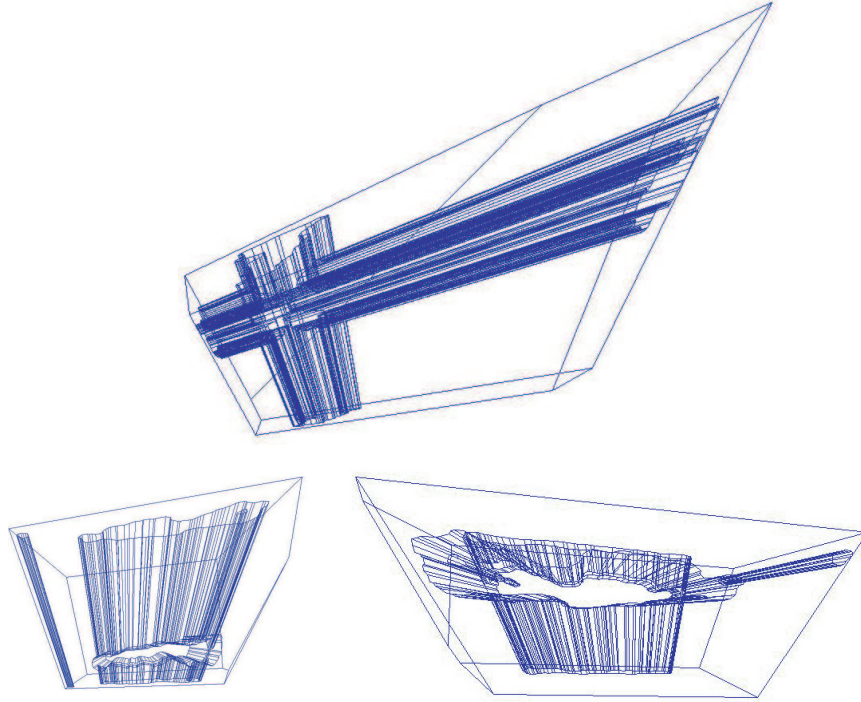
Figure 2-3: The intersection of extruded silhouettes forms the visual hull. Above, the cube-like volume is the intersection formed from the projections of the rectangular image plane of two cameras, and within the volume is the intersection of the extruded silhouettes of a person.

a set of reference views K. Each reference view $k \in K$ has a silhouette image of the scene and a camera orientation. For each view k, we can form a cone-like projected volume C_k. This volume has faces formed from rays starting at the camera's eye and passing through all the boundary points of the silhouette image. The actual shape of the object(s) in the scene must be contained in this projected volume. As we increase the number of reference views, we know that the object shape must be contained within all the projected volumes:

$$vhull_K = \bigcap_{k \in K} C_k \qquad (2.1)$$

As $K \rightarrow \infty$, $vhull_K$ approaches a shape known as the *visual hull*. The visual hull is a tight fit around the object and is known to contain the convex hull as well. It does not always equal the object because concave regions cannot be discerned through silhouette information alone.

Figure 2-3 shows how the projected volumes of silhouettes from two views intersects to

produce a visual hull at the center. As the number of views increases, the shape at the intersection is refined.

2.1.2 Voxel Approach, CSG

Previous methods for shape estimation aim for simplicity. The intersection of three-dimensional volumes as done through constructive solid geometry (CSG) is difficult to do in a robust manner. A more common technique is to build a quantized representation of the visual hull by volume carving. A volumetric grid of voxel cubes marks where the object is and where the object is not. A standard voxel-based algorithm is detailed in Figure 2-4.

---

1. Divide the viewing region into a volumetric grid of $N \times N \times N$ discrete voxels.
2. Classify each voxel v_i in space as an *inner* voxel.
3. For all voxels v_i, $1 < i < N^3$:

   (a) Project v_i onto all K views.
   (b) If v_i falls outside the silhouette of *any* view, classify it as an *outer* voxel.

4. The visual hull is composed of all the *inner* voxels.

Figure 2-4: Summary of a voxel-based algorithm.

The voxel-based algorithm runs in time proportional to the number of voxels N^3. Researchers have found ways to make voxel-based algorithms faster for real-time applications. German Cheung describes a sparse pixel occupancy test (SPOT) algorithm as well as ways to align visual hull points using color and geometric constraints [4].

2.1.3 Visual Hull Sampling vs. Polyhedral Reconstruction

The voxel-based method suffers from quantization artifacts. The volumetric grid does not allow proper rendering on detailed regions of the object and is inefficient for achieving higher resolution. For this reason, the idea of sampling the visual hull along desired viewing rays was introduced [16] [15] [18].

Image-based rendering (IBR) offers a practical alternative to previous methods. The input is a set of silhouette images, and a new image from a desired view is produced directly from these images. If the primary use for the visual hull is to produce new renderings, then a view ray sampling approach (IBVH) is best. As described in [18], for a desired view, for each pixel in the resulting image, the intersection of the corresponding viewing ray and the

visual hull is computed. The computation is done efficiently as silhouette-line intersections in 2D image planes. The time complexity of the algorithm is $O(kn^2)$ if we have n^2 pixels and k source views. The efficiency of IBVH allows the approach to be used in real-time systems as shown by Matusik, *et. al.* [16].

Although a sampling approach provides a rendering of a desired view, the actual geometric shape of the scene object may be useful. For example, for face and gait recognition from multiple views, a view independent 3D shape is useful for tracking purposes [26]. A virtual camera placed in front of the face improves face recognition if a full frontal view is not given. Likewise, a virtual side camera may improve gait recognition.

If the visual hull itself is not necessary, an approximate geometric proxy is valuable. Face recognition systems depend on 2D images to recognize faces. However, the actual face is a three-dimensional surface so these systems have difficulty with pose variation and varying illumination. In order to best model the face, it is important to consider the underlying geometry.

The visual hull gives us a view-independent representation. We may create virtual cameras and view the hull from any position. It is also a data representation of pose. If analyzing a video sequence of a moving person, we can potentially store the visual hull at each time frame and build a motion graph along with the recorded hull shape at each node.

The polyhedral visual hull (PVH) algorithm is a variant of IBVH. Unlike IBVH, the PVH algorithm calculates a 3D shape. The computation is done efficiently as silhouette-triangle intersections in 2D image planes. The running time of the algorithm is $O(k^2n^2)$ as described primarily in [14].

2.2   Camera Representation

Before a discussion of the PVH algorithm, the basics of computation involve camera representations. We process a set of silhouette images that lie on the image planes of different cameras. From the camera eye, the contours of these silhouettes project outward according to the camera parameters.

In perspective projection, Figure 2-5, a point M in the world projects onto the screen at pixel position m according to equation 2.2. This is the standard computer graphics
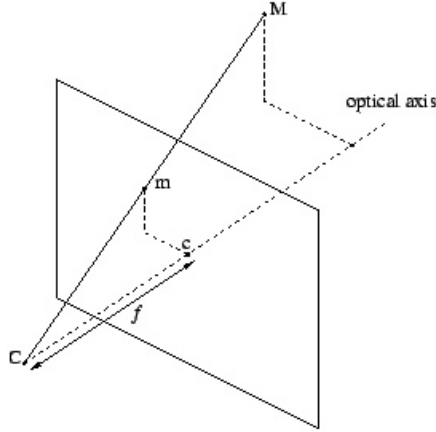
Figure 2-5: Perspective Projection.

equation.

$$m = \frac{1}{z} K [R \ t] M \tag{2.2}$$

The camera model includes matrices K for intrinsic parameters, matrix R for rotation, and vector t for translation. The intrinsic matrix K is written as:

$$K = \begin{bmatrix} \alpha & -\alpha \cot(\theta) & u_0 \\ 0 & \frac{\beta}{\sin(\theta)} & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The intrinsic parameters are comprised of α and β as the scale factors of the camera CCD, θ as the camera skew, and (u_0, v_0) as the center of the image. The extrinsic parameters include the rotation matrix R, a 3×3 orthogonal matrix specifying where to rotate the camera's look direction, and a vector t specifying where to translate the camera eye.

While equation 2.2 is simple enough, in vision applications the *inverse* formulation is usually required. We wish to translate pixel coordinates to world coordinates. In the PVH algorithm, we want to find the world coordinates of a contour pixel point $m = [p_x, p_y, 1]^T$. The equation is:

$$M = R^T K^{-1} m - R^T t \tag{2.3}$$

20

Knowing where the camera eye lies ($cam_{eye} = -R^T t$) and also where a particular contour point lies in world coordinates, we know the ray projecting from the eye through the contour pixel point.

Other camera representations are used primarily by the OpenGL graphics community. The OpenGL environment defines a *look* and *up* vector for camera gaze direction similar to rotation matrix R. For intrinsic specifications, the field of view in the y direction determines how far the image plane is from the camera eye. An aspect ratio marks the size of rectangular pixels, and a resolution specifies the number of pixels in the grid.

An alternative camera representation suitable for the PVH algorithm is shown in Figure 2-6. In order to find the world coordinates of a contour point $m = (p_x, p_y)$ we calculate: $M = camera_{eye} + \vec{z} + (p_x)\vec{x} + (p_y)\vec{y}$.
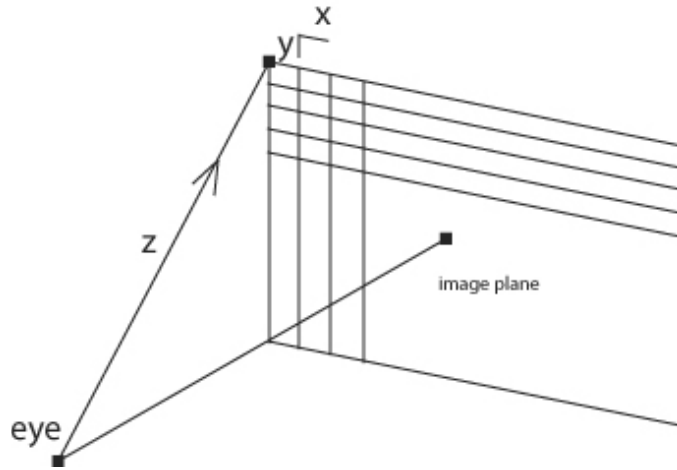
Figure 2-6: Camera representation with $\vec{z}$, $\vec{x}$, $\vec{y}$ vectors.

Converting camera parameters from one representation to another is often necessary. For example, if given K, R, and t, we convert to a $\vec{z}$, $\vec{x}$, $\vec{y}$ notation by calculating where the top left screen corner lies in world coordinates (using eqn 2.3) and subtract the camera eye to determine $\vec{z}$. Vectors $\vec{x}$ and $\vec{y}$ are computed similarly. For the PVH implementation, all three camera representations were necessary at different stages.

2.3 Silhouette and Contour Input

A basic processing step in many vision applications is background subtraction. For the PVH algorithm, we obtain binary silhouettes after subtracting out the background from raw

color images. In order to simplify segmentation, we used white sheets for background and saturated the cameras such that shadows were not a problem. The resulting background subtraction consisted of taking a difference between a scene with no person and a scene with a moving person along with filters for smoothing. A more complex segmentation algorithm could have modelled mean and variance at each pixel to distinguish foreground and background. In general, the quality of silhouettes is important since an arbitrary hole in just one silhouette will cause a hole through the visual hull.

In Figure 2-7, the silhouette input for building one visual hull is shown. The silhouettes are different views taken simultaneously. Noise and stray patches are eliminated from the final visual hull computation.

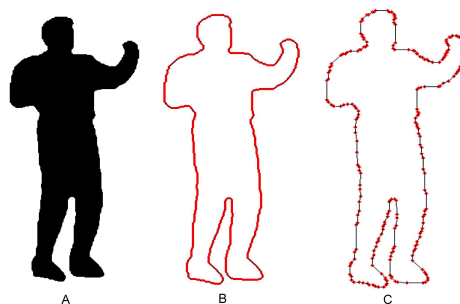Figure 2-7: Eight silhouette images captured simultaneously.

Figure 2-8: Contour approximation of a silhouette.

Each silhouette s is specified as multiple boundary contours that are connected chains of pixels. In order to efficiently represent a pixel chain, we can compress the number of points on the contour. For example, if we have five pixels on the same raster line, we only need to store the first and last points. This transformation is shown in Figure 2-8. Contours as pixel chains (B) are extracted from binary silhouettes (A) and approximated with a finite number of edges (C). Please refer to Chapter 3 for contour approximation.

2.4 Polyhedral VH Construction

As mentioned, the visual hull is the intersection of cones that are extruded silhouettes. The pre-processed input to the algorithm is a set of silhouette contours represented as polygons with edges. When these polygons are projected as cones from the camera eye, each edge of a polygon projects as one (infinite) *face* of the visual hull. When a projected face intersects with all other cone projections, the result is a finite, constrained planar surface set that lies on the boundary of the visual hull 3D structure. Once we calculate all boundary surface sets, we have defined the visual hull.

Figure 2-9 shows one face f of a cube silhouette. Face f extends infinitely; the purpose of the PVH algorithm is to find what regions along the face lie on the visual hull surface. In order to find these regions, we compute the projections of f onto all other image planes, calculate 2D intersections with silhouettes to get polygon sets, and then lift the polygon sets back onto the face. This process results in $K - 1$ polygon sets for face f if given K source views. The union of these polygon sets (a 2D operation) in the plane of f defines the surface that lies on the visual hull boundary.

Repeating the process for each projected face of each silhouette yields a collection of surface patches that define the visual hull. The upcoming sections provide further details.

2.5 Epipolar Geometry

The visual hull concept is based on epipolar geometry and the constraints enforced by multiple synchronous views of an object. Intuitively, if two cameras cam_1 and cam_2 capture silhouette information of a scene object, then we should be able to derive extra information about the scene if we know the relative positions of the cameras. For example, with only cam_1, we do not know the distance of any object from the camera. With two cameras, we
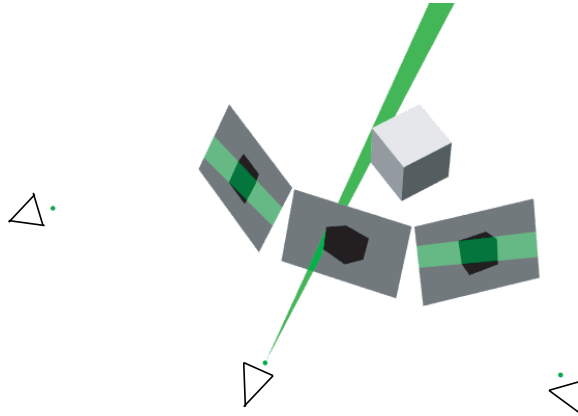
Figure 2-9: A single silhouette cone face is shown, defined by the edge in the center silhouette. Its projection in two other silhouettes is also shown [14].

can compute this distance - which is a principle of depth from stereo.

When given multiple source views, we can derive even more information. For a projected face of a silhouette, one camera by itself suggests that the face of the visual hull is infinite, i.e. there is no knowledge of distance. With two cameras, we know that the face is bounded. With K views, we are able to *refine* our estimate of where the visual hull face lies in space.

A cone projection from a silhouette is comprised of rays from the camera eye through pixel points on the silhouette contour. These rays are *contour rays*. If a silhouette contour has n edges, then there will be n contour rays and n faces on the cone. Figure 2-10 illustrates how contour rays from one camera are *epipolar rays* on the image plane of another camera.
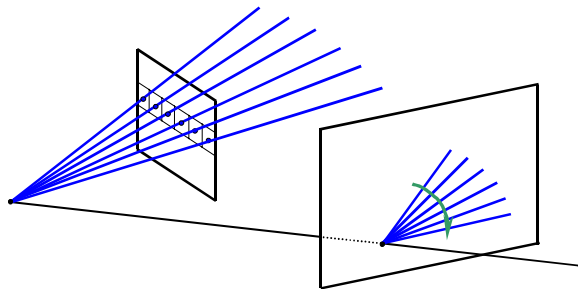
Figure 2-10: Rays from the camera eye through pixels in one image plane (contour rays) are seen as epipolar rays in another image plane.

In order to calculate the epipolar rays, we use epipolar geometry between two cameras, Figure 2-11. The epipole is the intersection of the line joining the optical centers of the cameras with an image plane. Line $\overline{OO'}$ intersects the image plane of the second camera at

epipole e'. The projection of contour ray $\overrightarrow{OX}$ from the first camera is the epipolar ray $\overrightarrow{e'x'}$ in the second camera's image plane. Alternatively, the plane defined by points O, O', and X cuts both image planes as lines $\overline{ex}$ and $\overline{e'x'}$. This defines the correspondence between both image planes. The epipolar constraint as stated in textbooks is merely that points along ray $\overrightarrow{OX}$ lie along the epipolar ray $\overrightarrow{e'x'}$.
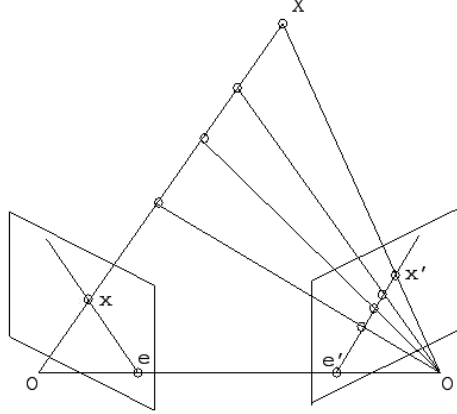
Figure 2-11: The epipolar line constraint.

Epipolar lines do not always intersect at the epipole and can be parallel to each other in degenerate configurations. For example, parallel camera planes that differ only in translation cause parallel epipolar lines where the epipole is at infinity. Our PVH algorithm requires that all pairs of cameras point towards each other. If they are parallel in any case, a small perturbation in any practical setup will avoid the singularity. Another special configuration is when two cameras point directly at each other. In this case, there exists some point X such that the epipolar plane defined by O, O', and X can not be defined because all points are collinear. To avoid this, we can take a point within a small epsilon radius about point X to define the plane properly.

2.6   Intersections in 2D

With epipolar geometry, we can relate one camera to another. For the visual hull, one cone face f is defined by two contour rays $\overrightarrow{c_1}$ and $\overrightarrow{c_2}$; these contour rays project as epipolar rays $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ on the image plane of another camera. In this section, we describe how to intersect the projection of face f with the silhouette on another camera's image plane. Through

epipolar geometry, we have reduced potential 3D intersections to simpler 2D intersections.

In order to efficiently perform 2D intersections, we use edge-bin data structures [14]. The edge-bin data structure splits a silhouette into angular regions extending from the epipole, Figure 2-12. A bin is defined by its start angle, end angle, and the set of edges that span its space. The angles are marked according to some arbitrary reference zero and 90° axes. The edge-bin table for our example is shown in Table 2.1.
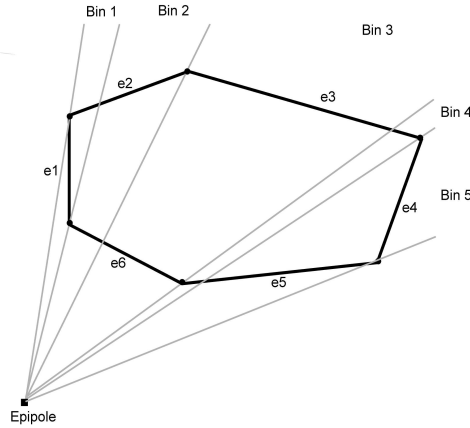
Figure 2-12: A bin structure records all edges that span its space.

Bin #	Edges
1	e_1, e_2
2	e_2, e_6
3	e_3, e_6
4	e_3, e_5
5	e_4, e_5

Table 2.1: Edge-Bin table.

With an edge-bin table, intersecting the projection of one face with a silhouette consists of finding the epipolar rays of the face and finding the start and end bins for these rays based on angle. Each epipolar ray will pass through one bin and intersect only the edges within that bin.

Figure 2-13 illustrates the efficient 2D intersection of a projected face defined by two epipolar rays with a silhouette contour. The gray region of intersection is a polygon set defined by its boundary points. How exactly is an intersection formed? The process requires edge-bin information, silhouette contour connectivity, and a final step of polygon

26

Figure 2-13: The triangle formed by the epipole and epipolar rays $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ intersects the silhouette contour. The intersection is the gray region.

compaction. Table 2.2 describes the steps taken to create and compact a polygon list.

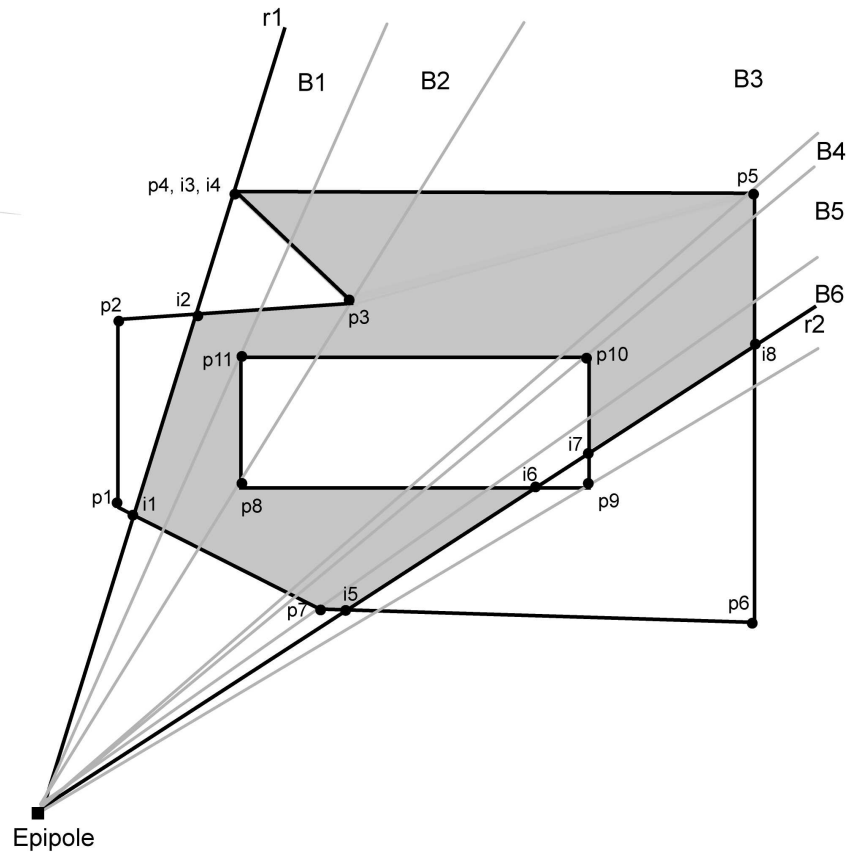| Step	Description	Lists of Polygons
1	Add intersection points	$[i_1, i_2]\ [i_3, i_4]\ [i_8, i_7]\ [i_6, i_5]$
2	Add p_{11}	$[i_1, i_2]\ [i_3, i_4]\ [i_8, i_7]\ [i_6, i_5]\ [p_{11}]$
3	Add p_8, p3	$[i_1, i_2, p_3]\ [i_3, i_4]\ [i_8, i_7]\ [i_6, i_5]\ [p_8, p_{11}]$
4	Add p_5	$[i_1, i_2, p_3]\ [p_5, i_3, i_4]\ [i_8, i_7]\ [i_6, i_5]\ [p_8, p_{11}]$
5	Add p_{10}	$[i_1, i_2, p_3]\ [p_5, i_3, i_4]\ [i_8, i_7, p_{10}]\ [i_6, i_5]\ [p_8, p_{11}]$
6	Add p_7	$[p_7, i_1, i_2, p_3]\ [p_5, i_3, i_4]\ [i_8, i_7, p_{10}]\ [i_6, i_5]\ [p_8, p_{11}]$
7	Begin Compaction	$[p_5, i_3, i_4, p_3, i_2, i_1, p_7]\ [i_8, i_7, p_{10}]\ [i_6, i_5]\ [p_8, p_{11}]$
8		$[p_7, i_1, i_2, p_3, i_4, i_3, p_5, i_8, i_7, p_{10}]\ [i_6, i_5]\ [p_8, p_{11}]$
9		$[i_6, i_5, p_7, i_1, i_2, p_3, i_4, i_3, p_5, i_8, i_7, p_{10}]\ [p_8, p_{11}]$
10		$[p_{10}, i_7, i_8, p_5, i_3, i_4, p_3, i_2, i_1, p_7, i_5, i_6, p_8, p_{11}]$

Table 2.2: Finding the points of the intersection polygon.

The first step is to find intersection points $i_k, 1 \leq k \leq 8$. These points are the intersections of the epipolar rays $\overrightarrow{r_1}$ through the boundary of Bin 1 and $\overrightarrow{r_2}$ through Bin 6. Note that there are two intersection points i_3 and i_4 that are identical. The reason is that i_3 belongs to edge $\overline{p_3 p_4}$ while i_4 belongs to edge $\overline{p_4 p_5}$. Ray $\overrightarrow{r_1}$ intersects both edges and the duplicate points record different connectivity data. All intersection points come as pairs except when the epipole is within the silhouette contour; in this case, we add the epipole to the polygon list twice. For example, $[i_1, i_2]$ and $[i_3, i_4]$ are regions of ray $\overrightarrow{r_1}$ that are considered *inside* the silhouette. Segment $[i_2, i_3]$ is outside the silhouette. Likewise, as we trace the intersection polygon in the other direction, $[i_8, i_7]$ and $[i_6, i_5]$ are also inside the silhouette.

Steps 2 to 6 consist of adding points along each interior bin using the edge-bin data structures. In step 2, we add p_{11} which is the point defining Bin 2. In step 3, we add points p_8 and p_3 - points at the same angle that define the beginning of Bin 3. In order to insert these interior points, we search over all sublists in our polygon list and check whether the interior point may be added at the front or end of the sublist. Since p_{11} does not fit in any sublist, it becomes its own new sublist. Eventually, the sublists will compact together as more interior points are added.

In step 7 we begin the compaction of sublists after all interior points from Bin 2 to Bin 6 have been added. We try to merge the first two sublists at the endpoints. From connectivity data, we know that sublists $[p_7, i_1, i_2, p_3]$ and $[p_5, i_3, i_4]$ have a connection through i_4 and

p_3. We continue compacting sublists until no more compactions are possible. At this point, we can remove any duplicate points if desired. In the example, by step 10 we have formed one large polygon list that describes the intersection polygon. Sometimes, we are left with two or three separate polygon lists.

The polygon formation and compaction scheme succeeds for general configurations. Figure 2-14 is a panel of cases encountered. Input silhouettes, especially silhouettes of people, may be concave polygonal contours. They may contain holes and nested contours. Multiple objects in a scene will also produce multiple contours.
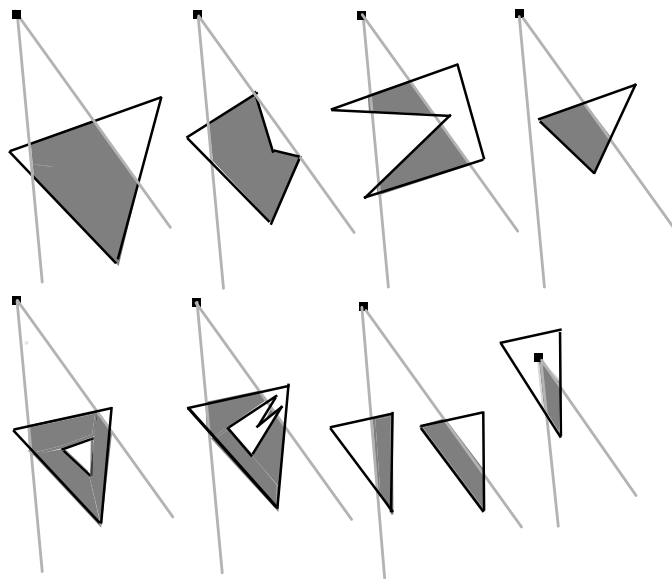
Figure 2-14: Many possible configurations exist for 2D intersection of the epipolar rays with the image contour: simple convex, concave, split intersection, border case, hole, nested contour, multiple contours, and interior epipole.

2.7   Visual Hull Faces

After the 2D intersection stage, each face f of a cone has an associated polygon set from its intersection with the silhouette from another view. We now lift the polygon set back onto the face so that the face is bounded and represents one surface patch of the visual hull, Figure 2-15.

Once we generate one surface patch, we can generate all surface patches on all cone faces. With only two cameras, we are able to output a visual hull volume as a simple
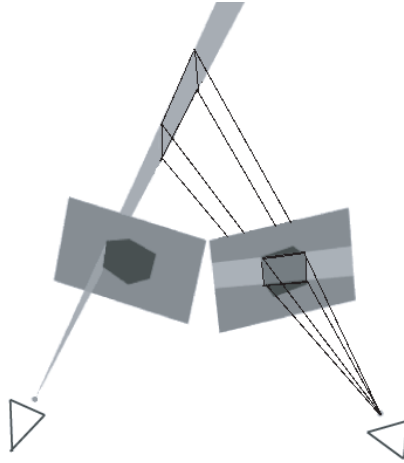
Figure 2-15: A single cone face is shown to extend infinitely. After finding the intersection of its projection on another image plane and a silhouette, we lift the intersection back onto the face in order to bound the visual hull surface.

approximation to our scene object(s). In Figure 2-16, the two camera composition of a person is a box-like visual hull. Refinement of shape is not possible with two cameras.
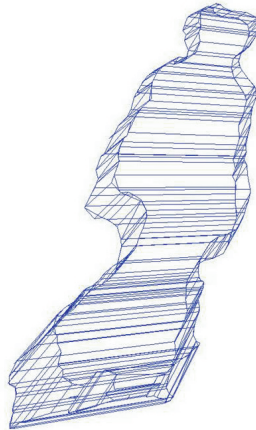
Figure 2-16: Two camera composition.

2.8   Refinement

With $K > 2$ views, we would expect the shape to contain more faces with a greater degree of roundness. In the limit, as the patches shrink in surface area, the visual hull should resemble a continuous surface volume. For K camera composition, we find the intersection

of the projection of a face f with the silhouettes of all other $K - 1$ views. This results in $K - 1$ polygon sets for each face. A refined surface patch of the visual hull is the union of these polygon sets in the plane of the face, Figure 2-17.
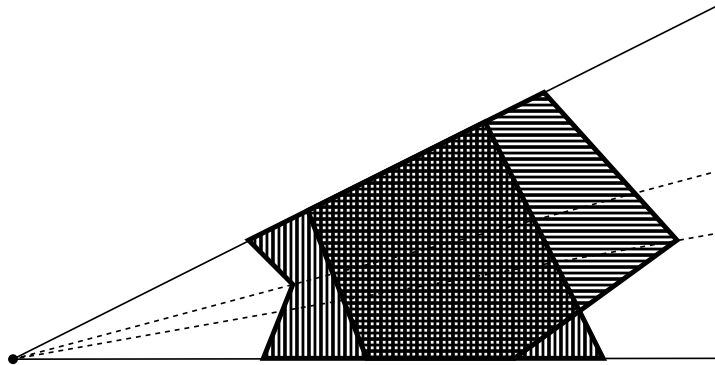
Figure 2-17: The union of polygon sets on the plane of a face (shown as the double hatched region) is the new refined surface patch of the visual hull [14]. Above, we intersect two polygon sets.

How do we find the union of $K - 1$ polygon sets? This subproblem is similar to finding the intersection of the epipolar rays with a silhouette using edge-bin data structures. To employ the same concept, we could decompose the polygon sets into quadrilaterals by dividing the space occupied by the polygons into triangular regions based on the polygon vertices and the apex of the silhouette cone [14].

Clipping 2D polygons is a basic routine in computer graphics and the operation is executed many times for rendering. In our implementation, we chose to use a 2D polygon clipping library that applies Vatti's polygon clipping method [30] [20]. In order to use the library, polygon sets described by 3D points on a plane for our system were reassigned 2D coordinates based on arbitrary reference axes on the plane. We also break the union of polygons sets into triangle strips for efficient rendering with graphics hardware.

Once we find the union of polygon sets, we can add as many views as desired for visual hull computation. Each view contributes a set of refined surface patches to the final visual hull. Figure 2-18 shows the $K = 8$ camera composition of a rock object.

As we refine the visual hull, the complexity of the output mesh increases. If we have K source views, then we are performing $K(K - 1)$ pairwise refinements. If we increase the number of source views by one, then we will have $2K$ extra pairwise refinements - a noticeable improvement especially if the views are orthogonal. Figure 2-19 is a panel of
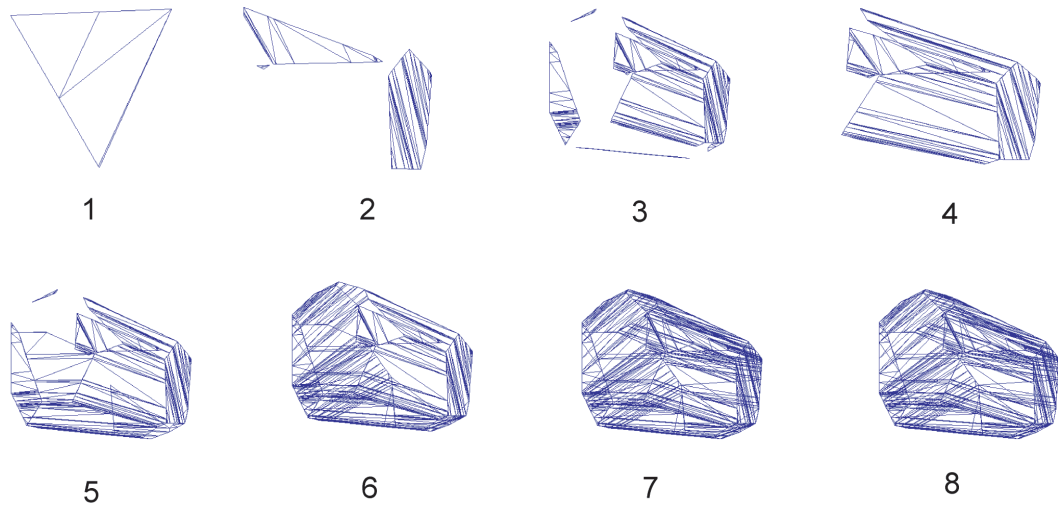
Figure 2-18: With $K = 8$ views, each view contributes a surface set of triangles to the final mesh. Drawings 1-8 show how new surface triangles from each added view comprise the final mesh.

visual hull constructions from $K = 2$ to $K = 8$ views. With eight views, the visual hull often contains up to $20,000$ triangles even after input contours are compressed. The output mesh is obviously not a regular mesh. It may contain T-junctions and the triangles may be long and thin.

2.9 Summary

In this chapter, we have detailed the entire PVH algorithm as implemented in a new system to acquire shape from silhouettes. A summary of the algorithm is given in Figure 2-20. While there are many available implementations of image based visual hulls, there are few easy-to-use versions of the polyhedral visual hull because of its computational complexity. To engineer the system from scratch takes time. Nevertheless, excitement for visual hulls and image-based rendering has propelled researchers to experiment with constructing higher-quality visual hulls of both types.

One way to improve the polyhedral visual hull is to add surface details and concave regions. In ordinary PVH reconstruction, silhouettes fail to capture crevices and indentations because there is no concept of depth over a silhouette. If we combine depth-from-stereo and visual hull reconstruction, we can improve rendering quality [13]. Another idea would be to use color consistency for rendering views. With color constraints, Image Based Photo

Hulls are tighter approximations contained within the visual hull [27]. Lastly, the authors of the polyhedral visual hull (Matusik et. al.) have used an alpha channel representation to construct Opacity Hulls [17]. Opacity hulls model highly specular and fuzzy materials such as fur and feathers. They are able to model complex objects that are impossible to scan with traditional scanners.

In conclusion, the PVH algorithm is a useful black box for acquiring shape from silhouettes. In recent years, papers have been written to add details to the shape. From a graphics standpoint, image-based rendering and data driven modelling have become a revolution. From a vision standpoint, a visual hull is a representation of pose and can be used to understand human motion patterns as explored in future chapters (Chapter 5).

6304   8799   11,112

12,991   16,548   17,890

19,211
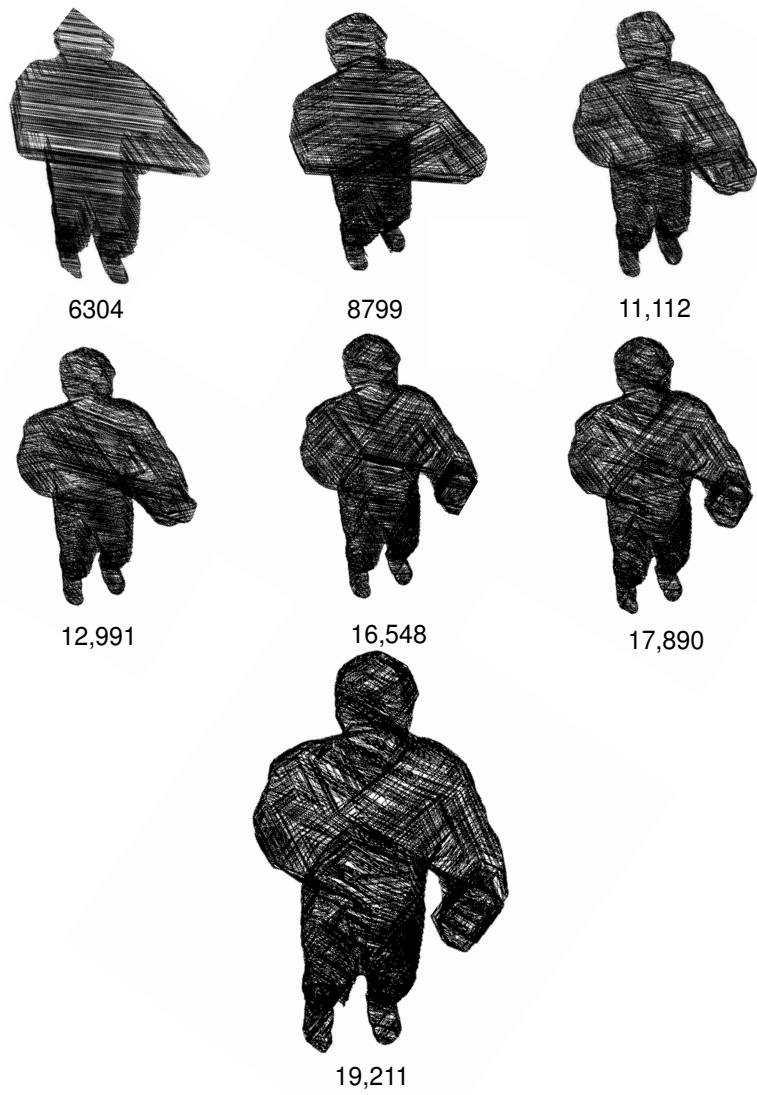
Figure 2-19: Visual hull refinement with K = 2 to K = 8 cameras. The number below each image corresponds to the number of triangles in the triangle mesh.

1. For all views $V_i, 1 < i < K$:

   (a) For all cone faces f of view V_i:

       i. Intersect the epipolar projection of face f with the silhouettes on the image planes of all other views V_j where $j \neq i$ using edge-bin data structures.
       ii. Lift each intersection polygon set back onto face f for a total of $K - 1$ polygon sets.
       iii. Find the union of the $K - 1$ polygon sets using polygon clipping. The union of the polygons is one surface patch of the visual hull.

   (b) Store all surface patches along each face f from view V_i.

2. The visual hull is the volume enclosed by all the surface patches from all views.

Figure 2-20: Summary of a polyhedral visual hull algorithm.

Chapter 3

Compressing and Processing Visual Hulls

The challenge of producing sequences of visual hulls includes the efficient computation of many video frames, as well as processing irregular output meshes. This section describes how to compress the output mesh by compressing 2D input contours using wavelet techniques. In addition, subdivision surfaces address the problem of irregularity in the visual hull mesh. After mapping a hierarchical subdivision surface onto the hull, we can manipulate the mesh properly and use interpolation schemes like Butterfly subdivision to smooth the mesh. The focus in both 2D and 3D cases is to first generate an underlying *coarse approximation* to a complex structure and then refine the approximation by adding details. Multi-resolution mesh approximation is a main subject in recent research, as described below in the text.

3.1   VH Computation

Visual hull shape complexity in terms of size and number of triangles depends on the number of edges in each of the input silhouettes. If we have input contours with too many edges, we will produce a hull with more triangles but with little improvement in shape accuracy.

As shown in Figure 3-1, at each frame of video for our system we must calculate a visual hull from 8 silhouettes. For a video of 1 minute and 30 seconds, as many as 1500 visual hulls are produced. We would like to compress the input contours to speed this computation and process the output meshes in some standard format.

Visual hull computation allows compression of both 2D contour inputs and 3D output

Figure 3-1: At each frame i, silhouette images produce a polyhedral triangle mesh.

meshes. For the 2D case, we would like to approximate the contours with as few pixels as possible in order to have few edges while at the same time capturing the 2D shape accurately. In the 3D case, the output mesh is irregular with thin triangles potentially causing sampling problems. The VH mesh is not adequate for multi-resolution analysis or interpolation schemes. We would like to map a different representation onto the output mesh to make it easier to process.

3.2 Multi-resolution of Contours

The 2D contour approximation problem is to take an input contour with many boundary points and produce a contour with similar shape but with less boundary points. For an image on a discrete pixel grid, we are given a set of N connected pixels, i.e. $(p_1, p_2, p_3, p_4, ...p_N)$, where two adjacent pixels represent one edge of a contour. We want to represent the contour shape with $n < N$ pixels. The approximating shape with n boundary pixels should include details where necessary (for example, places of high curvature), and less detail in flatter regions of the contour.

One way to approximate contours is to choose dominant points along the contour and minimize an error criterion. Another way is to use wavelet analysis for the multi-scale resolution of shape contours [23]. We test this approach. Although the approximating shape may not have the exact area as the original contour, it captures the shape accurately. For visual hull purposes a wavelet approach suffices because the silhouettes are noisy from background subtraction from the beginning. For general pose inference using contour matching (Chapter 5), we need to retain shape details; wavelets offer a way to add details at desired

38

scales.

In approximating contours, the first step is to parameterize the shape according to point number and separate the contour into X and Y coordinate functions, Figure 3-2. The
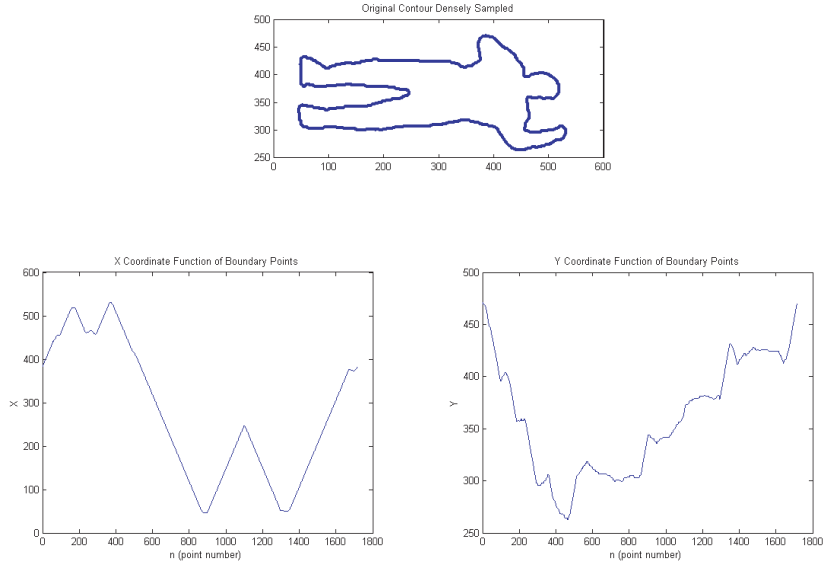
Figure 3-2: Split original input contour (1716 points) into coordinate functions.

split creates a correspondence between the X and Y 1D signals. The 1D signals describe discrete contours on image planes so the coordinate functions are piece-wise constant over intervals based on point number. For continuous contours, we would either need equations to characterize the contour or a dense sampling of points.

Given piece-wise constant coordinate functions, the wavelet to use for decomposition is the Haar orthogonal wavelet. We would like to approximate piece-wise constant functions at different scales as given through eqn 3.1. In most contexts, the Haar wavelet is not adequate. For example, if smoothing a surface, a filter of longer length such as the Daubechies 9/7 tap would work better. Haar averaging is not ideal for image compression as well since reconstruction is not smooth to the eye as with B-spline filters.

$$f(t) = \sum a(k)\phi(2^j t - k) = \sum c(k)\phi(2^{j-1}t - k) + \sum d(k)w(2^{j-1}t - k) \qquad (3.1)$$

For our person contours, using a six level decomposition, a coarse approximation (see Figure 3-3) is possible by taking only the coarse level coefficients and zeroing all detail

39

coefficients. With plain down sampling, the coarse contour is only 27 of 1716 points.
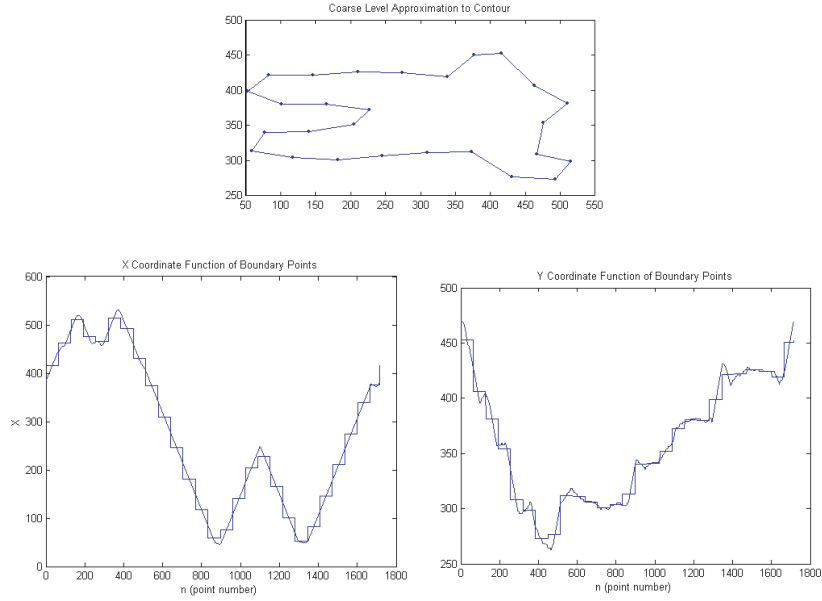
Figure 3-3: Coarse contour approximation. The compressed contour has 27 points as opposed to the original 1716 points. Bottom panels describe X and Y coordinate functions with their approximations.

In order to obtain finer approximations to the course shape, we would like to increase detail at desired scales. We add back detail coefficients (eqn 3.2) according to a span criterion and reconstruct the contour both in X and Y.

$$DWT(X_i) = (c_0, d_1, d_2, d_3, ...d_L). \tag{3.2}$$

The span criterion is shown in Figure 3-4. If the ratio of the perpendicular maximum deviation of the contour curve to the length between discrete points is above a user defined threshold, then the contour deserves to be approximated at the next finer scale. The 2^j factor adjusts for varying scale levels.

With a suitable threshold, we can find a suitable finer approximation to the coarse contour, Figure 3-5. By varying the threshold, we achieve different resolutions of the original contour.

As demonstrated in Figure 3-6, even with a 100 points, we can achieve a reasonable approximation for visual hull calculations because we include more points at the scales that

40

SPAN CRITERION

max_dev

length

$$max\_dev / length > 2^j \, T$$
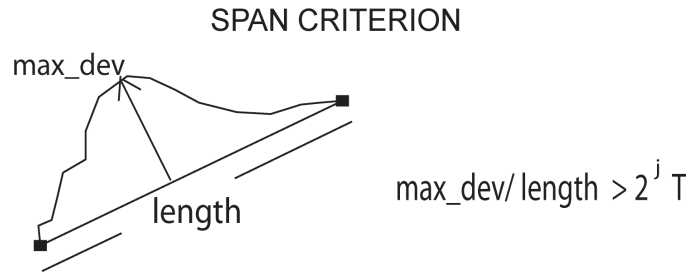
Figure 3-4: Criteria for finer approximations [23].
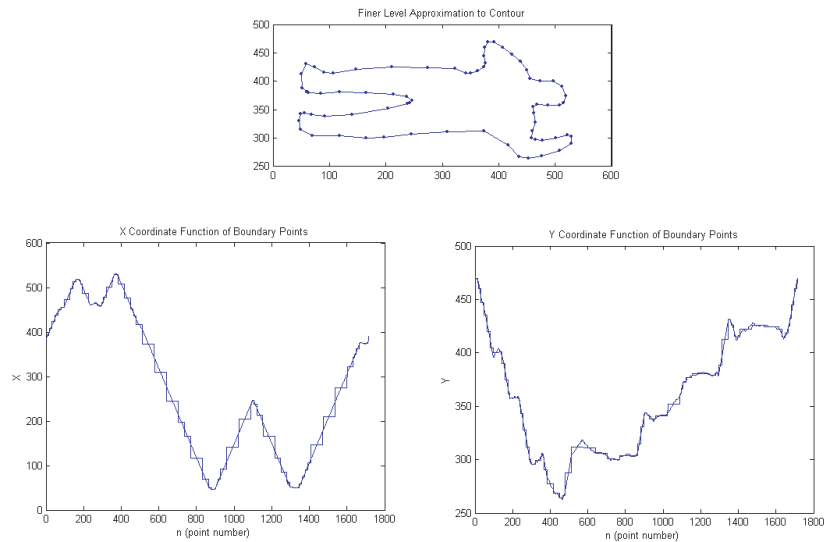
Figure 3-5: Fine contour approximation with 74 points as opposed to 1716 points. Bottom panels show finer approximation to the coordinate functions.

need more detail such as the arm or hands. Without any compression, if input contours contain approximately 1700 boundary points, the visual hull mesh contains up to 80,000 triangles. With compression to only 200 points per input contour, the mesh contains about 15,000 triangles and does not lose much in shape detail.

3.3 Subdivision Surfaces

In the past section, we saw how to facilitate visual hull computation by wavelet-encoding 2D input contours. Compression and processing of meshes in the 3D domain is more challenging. For example, consider the simple case of finding normals for all triangles in
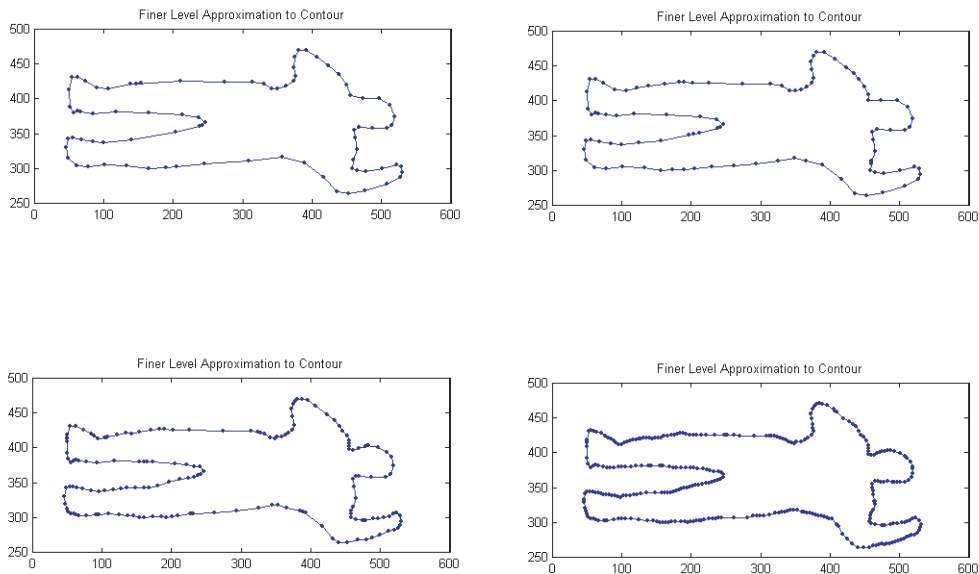
Figure 3-6: Multi-resolution of shape contours. Proceeding from top left, to top right, bottom left to bottom right, we have contours of 74, 101, 171, and 330 points.

a mesh. It is simple to find the normal of one triangle alone via cross products, but with 1000 triangles, the normals could be facing inward or outward on the mesh for a total of 2^{1000} configurations. How do we determine which way each normal points? The solution is to orient one triangle on the mesh and *propagate* the orientation to nearby triangles using connectivity data.

The visual hull output mesh is an irregular mesh which may contain T-junctions and thin triangles, disrupting the notion of a hierarchical basis. In order to approximate the visual hull, some form of re-meshing through subdivision surfaces is necessary. Compression and simplification of 3D meshes is an active area of research, see [5] for a brief review of wavelets on irregular point sets.

One idea for representing meshes is through normals. A normal mesh is a multi-resolution mesh where each level can be written as a normal offset from a coarser version. Hence the mesh can be stored with a single float per vertex [10].

Our approach to simplification shown in Figure 3-7 is through quaternary subdivision.

42

A tetrahedron is embedded in the irregular visual hull mesh and subdivided such that each face of the tetrahedron forms 4 new triangles. The triangles are lifted according to the distance away from the hull mesh and may be contracted backwards as well. This limiting subdivision process yields a series of approximations to the true 3D mesh $S^1, S^2, \ldots S^p$ where surface S^p contains 4^p triangles. The subdivision scheme with a base tetrahedron will work for a face mesh or any reasonable surface. It will fail for an entire human shape because of multiple parts. In this case, a coarse mesh other than a tetrahedron must be defined, and the subdivision process may still be applied.

Once a proper subdivision surface is defined, a Butterfly or Modified Butterfly scheme is useful for interpolating smoothly over the mesh [31]. From quaternary subdivision, the canonical case is a mesh vertex with valence 6 or six connecting edge lines. With increasing subdivision, boundary vertices and extraordinary vertices become isolated. When interpolating, instead of lifting vertices to a target mesh, a new vertex is based on weights of neighboring points. In the 1-dimensional case, this can be viewed as taking a point set and using a $[\frac{-1}{16}, \frac{9}{16}, \frac{9}{16}, \frac{-1}{16}]$ filter for smooth interpolation when inserting new points.

3.4   2D vs. 3D Approaches

Visual hull computation allows compression in both 2D and 3D domains. The benefit of compressing 2D contours is simplicity. We also eliminate redundancy in contour pixels while increasing computation speed. Wavelets offer the advantage of adding details at different scales. However, after computation, the visual hull mesh is still irregular. If we need to convert the mesh to a standard format with uniform triangles, subdivision in the 3D domain is necessary. Subdivision yields a mesh that is applicable for smoothing and interpolation schemes. Other parameterizations for re-meshing surfaces and multi-resolution definitions for meshes can be found in [24][9]. In general, re-meshing the VH requires post-processing time while wavelet-encoding of contours is fast.
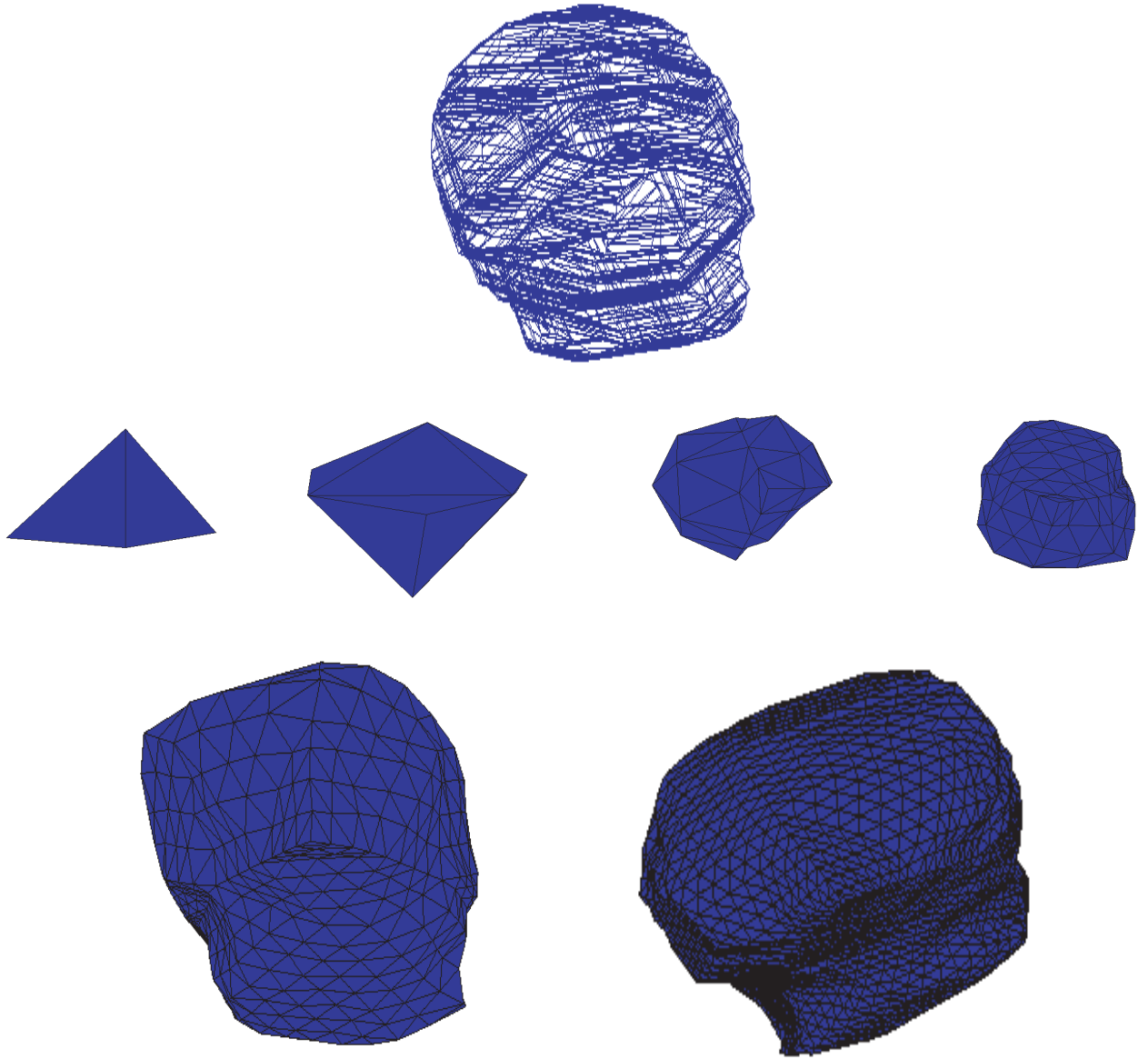
Figure 3-7: Subdivision surfaces for an irregular visual hull mesh. The visual hull mesh contains 1816 triangles. Each subdivision mesh contains 4^p triangles where p is the subdivision level. Level $p = 1$ corresponds to a base tetrahedron.

44

Chapter 4

Capturing Data Sequences

In previous chapters, we developed the visual hull as a means for acquiring shape from silhouettes and experimented with ways to process meshes. In this section, we describe how to capture data sequences to study the pose and motion of people. Multi-view data capture requires a well-constructed camera setup. The sharpness of visual hull results depends on **camera synchronization**, **camera placement**, and the accuracy of **multi-camera calibration**.

4.1 Synchronized Camera Setup

We constructed an eight-camera system based on a single PC station with 2.66 gigahertz processor speed and 4 GB of RAM, Figure 4-1. With three or four cameras, the visual hull is a rough geometric proxy, but with the number of views doubled to eight, the hull becomes a constrained realistic shape after pairwise refinements.

Camera synchronization in many scenarios is solved manually through a timer signal that initiates capture. For our purposes, we used DragonFly cameras from Point Grey Research and built infrastructure on top of their software platform [22]. The DragonFly cameras synchronize automatically when placed on one bus. The only other limiting factor was bandwidth so for eight cameras we used a sync unit to synchronize four cameras connected to one 1394 hub with four cameras on a second hub. Both outputs of the sync unit were connected to separate cards on a single PC.

After setting up the camera system, the main challenge was to capture and save images from eight cameras on one computer at 15 fps. At 15 fps, there exists 66.7 milliseconds of
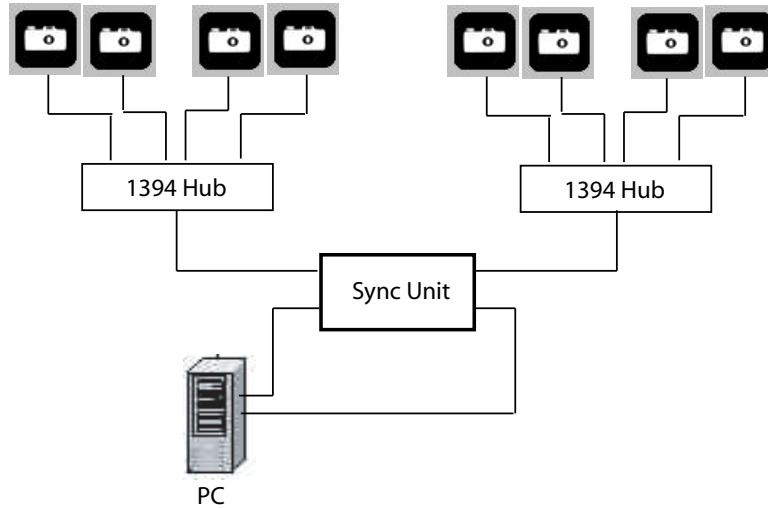
Figure 4-1: Synchronized Camera Setup.

processing time allotted for 1 frame. In this time, the task is to acquire and save 8 color images of size $640 \times 480 \times 3$ - a total of 7 mega-bytes of image data. Most systems that try to acquire this much data (105 mega-bytes per second) use a networking approach with multiple computers. This introduces further complexity in terms of network lag time and synchronization of multiple processors.

For our problem, the first observation was to defer the `save` operation until after data capture. With 4 GB of RAM or an effective 2 GB given to a program under Windows O/S, the goal was to keep all images in memory until capturing a full sequence of motion. However, without image compression, we would only be able to store approximately 30 seconds of video in memory.

To capture longer sequences, we implemented two steps. The first step was to capture gray scale images from the DragonFly cameras and reconstruct color through interpolation after data capture. The second step was to use Intel's Image JPEG Library to compress images in real time (see [21] for an article). With JPEG compression, even under compression ratios of 6:1, images do not lose much detail since most of the background is a uniform color.

Using efficient data structures, we were able to acquire up to 5 minutes of synchronous video depending on the image compression ratio. For our purposes of recording samples of motion, this was more than enough time to capture diverse sequences. Of course after data capture, our system requires an additional few minutes to interpolate color and save images

to disk.

4.2 Multi-Camera Calibration

A main challenge for data capture is multi-camera calibration. This subproblem is similar to calibrating a network of distributed sensors. Each camera has local information of the environment in the form of an image with detected feature points. We would like to determine the relative positions of all cameras in a global reference system based on local data.

reconstructed points/camera setup only inliers are used
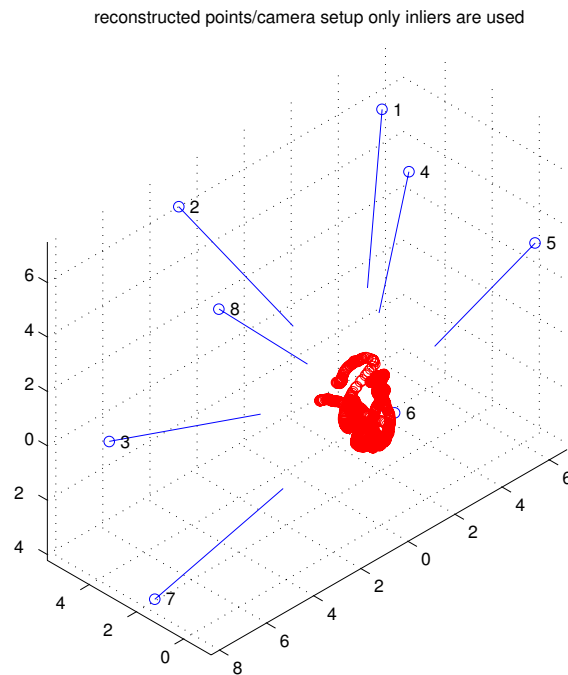
Figure 4-2: Multi-camera calibration.

Our camera placement is shown in Figure 4-2 which also shows the point cloud used to calibrate the cameras. Ideally, cameras should be placed in orthogonal configurations. If we place two cameras very close to each other as in stereo, the silhouettes are nearly identical and cause duplicate information. The optimal configuration for K cameras is a spherical placement, i.e. cameras spread along the surface of a sphere pointed towards the center. In practical situations, wall boundaries and a floor prevent us from placing cameras optimally.

A typical way to calibrate one camera is to take a picture of a calibration object such as a checkerboard pattern, find 2D features in the image and derive calibration from the 2D

features and their corresponding positions in 3D. With even 20 feature points, an accurate camera representation of both intrinsic and extrinsic parameters is possible. For multi-camera calibration, we could apply the same single camera calibration technique to all cameras. However, this would be tedious and would not take advantage of robust estimation across all cameras through a global error criterion for fitting feature points.

For our system, we extended a camera calibration toolbox built by Svoboda, et. al. [28]. We calibrate multiple cameras by waving a laser pointer in a dark room and by capturing a small 10 second sequence from all synchronized cameras. In the general case of m cameras and n captured points $\mathbf{X}_j = [X_j, Y_j, Z_j, 1]^T, j = 1 \ldots n$, we have n points projected on the image planes of each camera as 2D pixel points $\mathbf{u}_j^i$ where $i = 1 \ldots m$. The projection equation is described by

$$\lambda_j^i \begin{bmatrix} u_j^i \\ u_j^i \\ 1 \end{bmatrix} = \lambda_j^i \mathbf{u}_j^i = P^i \mathbf{X}_j$$

where each λ_j^i is a projective scale, each P^i is a 3×4 projection matrix and u, v are pixel coordinates [28]. Knowing only the observed pixel points $\mathbf{u}_j^i$ in each camera image plane, we would like to find each P^i and scale λ_j^i.

This problem is similar to a structure-from-motion algorithm such as the Tomasi-Kanade factorization algorithm for affine shape from motion [8]. All points and camera projections can be stacked together to form a matrix representation

$$W_s = P_{3m \times 4} X_{4 \times n}$$

where W_s is the data matrix (with scales), $P = [P^1 \ldots P^m]^T$ is the matrix for *projective motion*, and $X = [\mathbf{X}_1 \ldots \mathbf{X}_n]$ is the matrix for *projective shape*. With enough laser points $\mathbf{u}_j^i$ detected in each image plane for calibration, the matrix W_s can be factored into P and X using rank-4 factorization. A Euclidean upgrade and RANSAC analysis for excluding outliers is documented by Tomas Svoboda in [28].

Once we know the projection matrices P of each camera, we can extract the K, R, and t intrinsic and extrinsic information. For an eight camera system with 100 laser points, the calibration process yields camera positions up to an arbitrary scale. The cameras positions are relative to each other and may be aligned to a fixed world reference frame if required.

4.3 Sequence Results

Using the PVH algorithm described in Chapter 2 and mesh algorithms described in Chapter 3, we generated sequences of visual hull poses from our data sequences of human motion. Figure 4-3 is a panel of rendered poses from a sample upper-body motion sequence from eight cameras. The visual hulls are shaded in OpenGL and rendered from an arbitrary view.
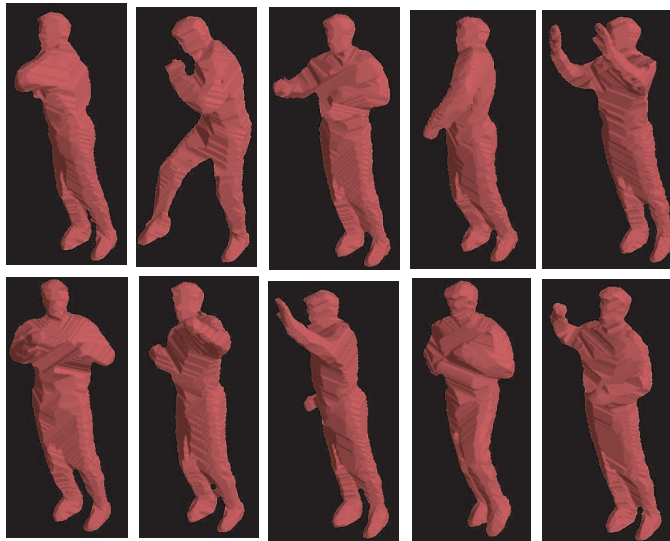
Figure 4-3: Rendered visual hull poses.

For our purposes of pose inference in Chapter 5, we do not need to texture the visual hulls. A simple texturing strategy for each hull might be to blend the original images per pixel to create surface texture. Let θ_i be the angle between the viewing ray of a virtual camera for a given pixel p, and the viewing ray of the i-th camera for p. Then each view is assigned a blending weight of $(1 - \frac{\theta_i}{\max_i \theta_i})$ and the value of p is a weighted sum of the pixel values in the original images [14]. Of course, this method does not account for visibility. View dependent texture mapping and the synthesis of color and texture information for virtual views is still an active area of research, see [6][19][7][2]. For high resolution acquisition and rendering, the original authors of the visual hull have modelled transparency and reflectance fields [17].

4.4   Data Capture Summary

Data capture consists of a series of steps including camera setup, camera synchronization, and camera calibration. These steps are not trivial. If an adaptive real-time calibration program for multiple cameras were developed, it would make the capture process much easier. Calibration also includes non-linear effects such as radial distortion of lens so linear models may not always suffice. Since the 8-point algorithm for computing the essential matrix, many papers have focused on deriving better calibration routines.

While it is time-consuming to capture real data, it would not be exciting to use synthetic sequences. Programs such as Poser model many different pose variations, but our sequences include natural transitions characteristic of actual behavior. We are also able to acquire shapes with deformable clothing such as skirts and robes that are difficult to model in Poser.

Chapter 5

Matching Sequences of Visual Hulls

After building sequences of visual hulls and capturing real data of human motion, the next step is to find patterns in pose. In a data driven approach, we would like to model the movement of a person based on sample input sequences. For example, given surveillance footage of a person walking for a short period of time from multiple cameras, we can capture their walking pattern. If given larger sequences of multi-view data, we have more knowledge of valid and realistic movements. To populate the entire space of human motions would be difficult, but a data-driven approach could still be useful.

From a graphics perspective, motion graphs already offer the ability to reanimate skeletons with realistic motion by finding patterns of motion. But, motion graphs do not specify shape and texture. Motion capture with markers requires expensive infrared cameras while shape from silhouette acquisition is relatively cheap. Based on these observations, the idea of combining motion graphs with visual hulls for shape and texture is appealing.

In this chapter, we introduce and test the idea of a *visual hull graph*, demonstrating how to match visual hulls across data sequences. A VH graph stores a view independent shape at each frame and the probability of transition between all pairs of frames based on similarity of pose. After building a graph from a sample sequence, the goal is to synthesize new video by finding cycles in the graph. We reduce the problem somewhat to make the data-driven approach more manageable. In order to match 3D poses, we use the original visual hull silhouettes from only one view to define a distance metric over 2D shape contours. More

complex matching might include skeleton extraction with angle and joint parameters.

The results obtained are encouraging to some extent, but they also point to new avenues for research and future work.

5.1 Previous Work

The idea of producing new video based on pose patterns is similar in nature to other problems in vision. For example, Efros and Leung produce various example textures from a single source texture based on local probability models [8]. In the same way as quilting the source texture will fail because of discontinuities at the edges, generating new video from source video by repeating frames randomly will fail because of abrupt transitions.

The idea of visual hull graphs is based in part on video textures [25]. Video textures are a form of media half-way between photographs and videos. Given an input sequence

Figure 5-1: Video Textures [25].

of images, a video texture is a video that is infinitely playable (i.e. can have arbitrary length) based on loops in the input sequence. Figure 5-1 shows the video texture of a candle flame. The candle flame loops over similar frames. If playing in the background, the candle motions give the illusion of no repetition because they are based on real source motions. The VH graph extends video textures because the visual hull is view-independent and we can synthesize new views at the same time as switching frames.

Lastly, motion graphs as previously mentioned are similar to visual hull motion graphs in the 3D domain. The sampling of kinematic poses from motion capture databases offers realistic and controllable motion [11]. Generating new motion is simply a walk through a motion graph. The VH graph maintains this simplicity but includes the extra information

of shape and texture at each frame.

5.2   Visual Hull Graphs

We generate a visual hull graph from a sequence of visual hull poses, Figure 5-2. In our data capture, we collected video with more than 1000 frames as source. Each frame includes a visual hull, the silhouettes for construction, and texture information if required.
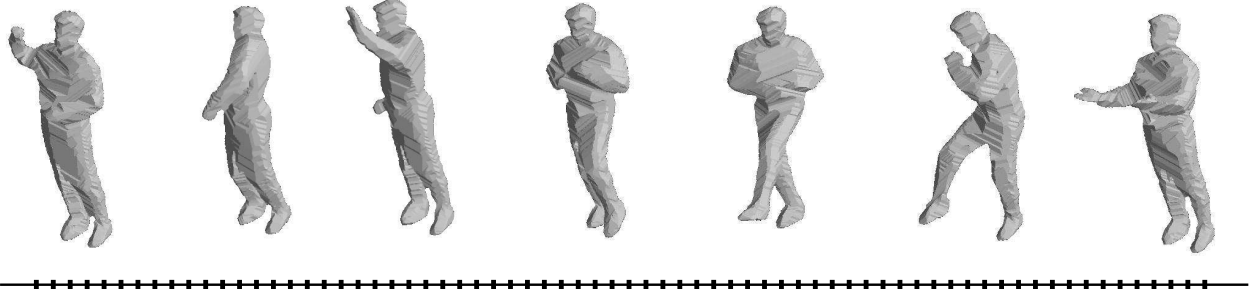

Figure 5-2: Building a graph from a sequence of visual hulls.

The graph representation is an adjacency matrix that represents the distance between all pairs of visual hulls according to a distance metric. To generate smooth transitions, we need to define a distance metric that will work for pose.

5.3   Shape Contour Matching

One way to obtain an adequate distance metric is to match 2D shape contours of silhouettes. To build our graph adjacency matrix, we only use the silhouettes from one view for matching. As shown in Figure 5-3, we match 3D visual hulls based on their projections as silhouettes on one stationary viewing plane. Obviously, this approach will not work in cases where single silhouette information fails - if for example we take a side view and a person moves his hand on the other side. However, a graph generated using single 2D shape contours worked well in practice. If pose ambiguities need to be resolved, multiple views could be used.

While matching 2D contours may appear simpler than matching 3D meshes, how do we match? In reality, this subproblem is challenging and must deal with translation, scale, and rotational invariance in the broadest sense. In our case, we analyzed simple body
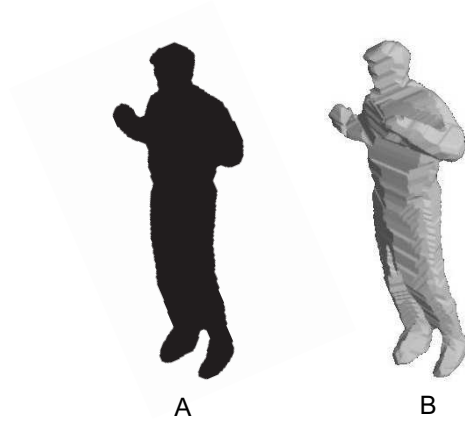
Figure 5-3: Inferring 3D pose (B) from a binary silhouette (A).

movements for one person at a time so matching did not require strict invariance to the above.

Distance metrics over 2D images or contours include the L2 norm, Chamfer and Hausdorff distances, and Earth-Mover's distance (EMD). Since our silhouettes were already in the form of boundary contour points, we chose to use Chamfer distances over contour sets, eqn 5.1.

$$D_{chamfer}(U, V) = \frac{1}{N} \sum_{u_i \in U} \min_{v_j \in V} \|u_i - v_j\| \tag{5.1}$$

Given two contour sets $U = \{u_i\}_{i=1}^{n}$ and $V = \{v_j\}_{j=1}^{m}$, the Chamfer distance function is the mean of the distances between each point $u_i \in U$ and its closest point in V [29]. If the Chamfer distance is five, then that implies that on average, a pixel on the first contour is five pixels away from the closest point to the second contour. We chose the symmetric Chamfer distance implemented efficiently through the Distance Transform of an image.

Given a metric, we can generate a small transition table using six consecutive silhouette images, table 5.1. The matrix of distances is symmetric and zero-diagonal with elements near the diagonal having lower chamfer distances because consecutive frames f_i and f_{i+1} are expected to be similar at 15 fps. In general, for visual hull graphs a Chamfer distance of less than seven pixels means that shapes are highly similar. Above that threshold, shapes in 2D do not generalize well to 3D pose and synthesized videos will be abrupt at the transitions.

0	2.51	4.7454	6.9481	10.2284	14.70078
2.51	0	2.3241	4.6289	8.0441	12.0151
4.7454	2.3241	0	2.4196	5.8560	9.8521
6.9481	4.6289	2.4196	0	3.5094	7.7002
10.2284	8.0441	5.8560	3.5094	0	4.4004
14.70078	12.0151	9.8521	7.7002	4.4004	0

Table 5.1: Similarity matrix for six contours of six consecutive frames. An element at position (i, j) is the Chamfer distance between contour i and contour j.

5.4 Transition and Probability Matrices

Finding the transition cost matrix using symmetric Chamfer distances for all pairs of frames results in $\frac{N^2 - N}{2}$ unique comparisons (excluding self comparisons) given N frames. The transition cost matrix is defined in eqn 5.2 and does not require much time to compute. For our matrix D_{ij}, we also disregarded all entries with a Chamfer distance above seven pixels.

$$D_{ij} = \|f_i - f_j\|_{chamfer} \tag{5.2}$$

In order to have smooth transitions, we would like the cost of transition to include whether local frames are also similar. For example, if we compare f_i with f_j, we would like to have f_{i-1} match with f_{j-1} as well as f_{i+1} with f_{j+1}. This filtering of the transition matrix is shown in eqn 5.3. The weights might be binomial, gaussian, or constant for averaging.

$$D'_{ij} = \sum_{k=-m}^{k=m-1} w_k D_{i+k,j+k} \tag{5.3}$$

The transition cost matrix is mapped to probabilities using eqn 5.4. If a particular (filtered) transition from f_i to f_j has a high cost, then the probability of transition should be low. The parameter σ controls the spread over transitions. With a smaller sigma, only low cost transitions are exercised, but a higher sigma allows for diverse transitions. In our case, we used a sigma value equivalent to a small multiple of the mean Chamfer distance across one row of the transition cost matrix. This ensured only smooth transitions. Another important observation for mapping probabilities is that we want to transition from frame f_i to frame f_j only if frame f_{i+1} is similar to frame f_j, not when f_i is close in distance to f_j. This

correction in generating new videos was important for smooth results.

$$P_{ij} \propto \exp^{\frac{-D'_{ij}}{\sigma}} \tag{5.4}$$

The last few equations, eqn 5.5 and eqn 5.6 add the notion of future costs of transitions. If we merely play random loops, eventually we will reach the end of the source video and not be able to transition back to an earlier frame. For this reason, if we add an additional cost for transitions toward the end, we will avoid a dead-end and loop out before advancing too far.

$$D''_{ij} = (D'_{ij})^p + \alpha \sum_k P''_{jk} D''_{jk} \tag{5.5}$$

$$P''_{ij} \propto \exp^{\frac{-D''_{ij}}{\sigma}} \tag{5.6}$$

For future costs, we include an α parameter similar to a Q-learning parameter in artificial intelligence literature [25]. We iterate between calculating the modified transition cost matrix D''_{ij} and the associated probability matrix P''_{ij}. D''_{ij} includes the filtered cost as before with a new term that is a sum over all future expected costs based on probability of transition.

Figure 5-4 shows the filtered transition matrix after a Chamfer distance threshold of seven for a source sequence of $N = 1342$ frames. Black regions indicate areas of low transition probability. From the figure, it is obvious that the visual hull graph will not contain many large repetitive loops but will have short loops as shown along the matrix diagonal.

5.5 Sequencing Loops

In order to sequence optimal loops of arbitrary length, one way is to generate all subsequences using dynamic programming as described in [25]. For our implementation, the cost for all legal loops is identical. Either the transition is legal and smooth, or we cannot use it for synthesis. A modified approach for constructing loops of arbitrary length was used.

Figure 5-5 shows simple loops such as $[A, B]$, $[C, D]$, and $[E, F]$ as well as a complex loop $[A, D]$. A complex loop consists of two or more overlapping loops. The goal of video synthesis is to generate a sequence of frames with loops or non-consecutive transitions. We
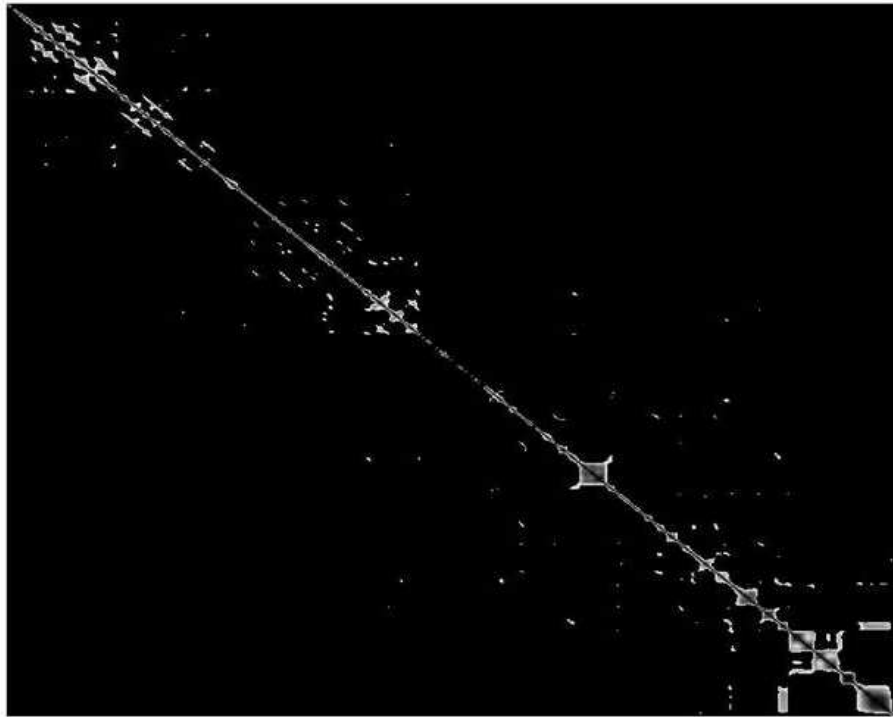
Figure 5-4: Transition Matrix.

can play loops randomly forever with future costs included, or link specific loops together for a video of desired length.

5.6 Results

From our motion sequences, we generated a few videos from our source to demonstrate the effectiveness of the visual hull graph. Transitions were smooth to the eye after filtering and thresholding. As expected from the transition matrices, our source video only included small loops. For example, in a repetitive punch sequence, our system could find loops between punches. However, over longer sequences, our system did not find matching poses.

The added benefit of using visual hulls is the ability to change viewpoint as well as frame. We generated additional videos with changing viewpoint to demonstrate the advantage of the VH graph representation.
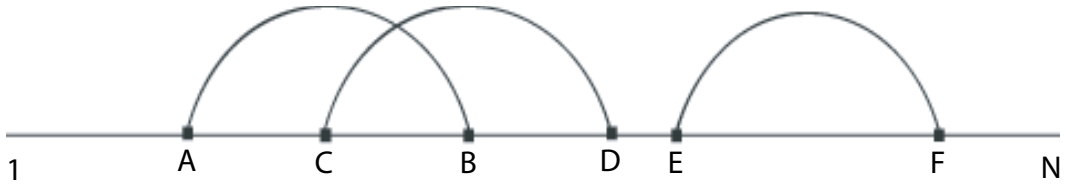
Figure 5-5: Simple and Complex Loops.

5.7 Conclusion

In conclusion, we note several challenges in data-driven modelling of motion:

1. First, a 3D distance metric for pose is required for generality although a 2D contour matching approach works to an extent for modelling single person motion. We could improve 2D shape matching by adding scale, translation and rotation invariance, or try to match 3D skeletons fitted to visual hulls.

2. The VH graph relies on the presence of loops in data. Our example graph did not contain large loops. We might need longer source video for more graph poses initially.

3. Ideally, we want to match poses between different people and transfer motion. We could build a graph based on one person's movement, and match another person's movement through the graph.

4. If poses will not match in a graph, interpolation of shape might be necessary. This problem is quite open to new ideas.

5. Controllable motion through a graph is also appealing although not explored fully in this chapter. With a visual hull graph, by adding extra parameters for position or velocity, we should be able to animate a shape to execute specific controlled actions.

The future work in this area is interesting because it is open to suggestion. Yet, through a limited context, visual hull graphs demonstrate how to capture patterns of motion.

Chapter 6

Discussion

Analyzing pose through a visual hull graph is an exciting research topic. The approach taken in Chapter 5 is a demonstration of what is possible through 2D shape matching, but it invokes a flood of new questions about data-driven human motion transfer. Ultimately, what would we like to accomplish?

From a vision standpoint, the main aim is to recognize and learn patterns of behavior. If we can track certain pose sequences, the problem of gesture recognition might seem easier to solve. From a graphics standpoint, visual hull graphs offer a way to animate 3D shapes. With shape and texture information at each frame, we can model the realistic movement of complex non-rigid objects.

While the big picture is important, often the subproblems encountered along the way require more thinking. In our study of matching and compressing sequences of visual hulls, we addressed numerous hard problems:

3D Shape Reconstruction. In Chapter 2, we introduced the polyhedral visual hull to acquire shape from silhouettes. Image-based modelling of dynamic scenes has been a revolution in computer graphics. The visual hull is one representation of shape, but it does not include surface details. It is also dependent on silhouette accuracy. If background segmentation creates holes in the silhouettes, then holes will appear in the hull. The problem of efficient and accurate 3D shape estimation is far from solved. Papers have combined traditional voxel carving algorithms with visual hulls, and recent research has used depth-from-stereo, color constraints, and opacity information to add more details to the 3D shape.

Mesh Compression. In Chapter 3, we introduced ways to compress the VH mesh

in both 2D and 3D domains. In 2D, we used wavelet-encoded contours to compress the visual hull for computational efficiency. 2D compression is possible until the contours lose shape consistency. In 3D, subdivision surfaces address the problem of irregularity in the VH mesh. If we want to apply existing mesh algorithms such as smoothing, we must re-mesh the VH shape first. Re-meshing algorithms and mesh multi-resolution are fresh areas of research. New 3D scanners are able to scan larger data sets and these data sets require proper visualization.

Camera Calibration and Data Capture. In Chapter 4, we described the problem of capturing sequences from multiple cameras. The main difficulty with multi-view data capture is to synchronize and calibrate all cameras. If we could automatically calibrate all cameras by tracking known points in the environment, our task would be simpler. The other problem of data capture is segmentation of foreground in images from background obscured by moving shadows and clutter. We solved the problem by using a controlled capture studio, but the problem of accurate image segmentation remains a challenge in most vision systems.

Shape contour matching. In Chapter 5, we used shape contour matching as a distance metric over 3D pose. Shape contour matching is useful in general for matching objects in images. The main problem is to develop a correspondence between two shapes. A Chamfer distance metric does not require any correspondence, but it lacks rotation, scale and translation invariance. If we solved fast and efficient shape matching, we could have a better image search on Google as well as better vision recognition systems for detecting objects in images.

Pose Inference. In Chapter 5, we also described ways to find patterns in pose through a visual hull graph. Other methods of pose inference might be to extract 3D skeletons from 3D shapes and match angle/limb parameters. This subproblem eventually leads towards the big picture of recognizing patterns of motion through data driven modelling - hence an area open to new ways of thinking.

To summarize, we explored many active topics of vision and graphics research through this project. We have produced a modular software system for the polyhedral visual hull and applied it in the case of visual hull graphs. The results are promising, but future work is inevitable!

Bibliography

[1] H. Baker, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, and T. Malzbender. The coliseum immersive teleconferencing system. In *Proc. International Workshop on Immersive Telepresence*, 2002.

[2] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *Siggraph*, pages 425–432, 2001.

[3] R. Yang C. S. Kurashima and A. Lastra. Combining approximate geometry with view-dependent texture-mapping - a hybrid approach to 3d video teleconferencing. In *SIBGRAPI, XV Brazilian Symposium on Computer Graphics and Image Processing*, 2002.

[4] (German) Kong-Man Cheung. *Visual Hull Construction, Alignment and Refinement for Human Kinematic Modeling, Motion Tracking and Rendering.* PhD dissertation, Carnegie Mellon University, 2003.

[5] I. Daubechies, I. Guskov, P. Schröder, and W. Sweldens. Wavelets on irregular point sets. *Phil. Trans. R. Soc. Lond. A*, 357(1760):2397–2413, 1999.

[6] P. Debevec, Y. Yu, and G. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Rendering Workshop*, 1998.

[7] A. Fitzgibbon, Y. Wexler, and A. Zisserman. Image-based rendering using image-based priors. In *9th IEEE International Conference on Computer Vision (ICCV '03)*, 2003.

[8] David A. Forsyth and Jean Ponce. *Computer Vision A Modern Approach.* Prentice Hall, 2003.

[9] I. Guskov, A. Khodakovsky, P. Schroder, and W. Sweldens. Hybrid meshes: Multires- olution using regular and irregular refinement. In *Proc. of SoCG*, 2002.

[10] I. Guskov, K. Vidimce, W. Sweldons, and P. Schroder. Normal meshes. In *Siggraph 2000*, pages 95–102, 2000.

[11] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *ACM Transactions on Graphics*, pages 473–482, 2002.

[12] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:150–162, 1994.

[13] Ming Li, Hartmut Schirmacher, Marcus Magnor, and Hans-Peter Seidel. Combining stero and visual hull information for on-line reconstruction and rendering of dynamic scenes. *Proc. IEEE International Workshop on Multimedia and Signal Processing (MMSP'02)*, 2002.

[14] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time render- ing. In *Proceedings of Twelfth Eurographics Workshop on Rendering*, pages 115–125, 2001.

[15] W. Matusik, C. Buehler, and L. McMillan. Efficient view-dependent sampling of visual hulls. In *MIT LCS Technical Memo 624*, 2002.

[16] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *Siggraph 2000*, pages 369–374, 2000.

[17] W. Matusik, H. Pfister, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3d photography using opacity hulls. In *ACM Siggraph*, 2002.

[18] Ẃojciech Matusik. Image-based visual hulls. Master's thesis, Massachusetts Institute of Technology, 2001.

[19] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Siggraph 95*, 1995.

[20] Alan Murta. A general polygon clipping library. Technical report, Univeristy of Manchester, UK, 1999.

[21] Mark Nelson. The intel jpeg library. Internet article available at: http://www.dogma.net/markn/articles/IntelJpgLibrary/, 2002.

[22] Point Grey Research. *FlyCapture User Manual and API Reference*, 2001.

[23] L. Prasad and R. Rao. Multiscale discretization of shape contours. In *Proceedings of SPIE*, pages 202–209, 2000.

[24] E. Praun, W. Sweldens, and P. Schroder. Consistent mesh parameterizations. In *Siggraph*, 2001.

[25] A. Schodl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Siggraph*, pages 489–498, 2000.

[26] G. Shakhnarovich, L. Lee, and T. Darrell. Integrated face and gait recognition from multiple views. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.

[27] G. Slabaugh, R. Schafer, and M. Hans. Image-based photo hulls. In *Proc. of 3D Data Processing, Visualization, and Transmission (3DPVT)*, 2002.

[28] Tomas Svoboda. Quick guide to multi-camera self-calibration. Technical report, Computer Vision Lab, Swiss Federal Institute of Technology, Zrich, 2003.

[29] A. Thayananthan, B. Stenger, P. H. S. Torr, and R. Cipolla. Shape context and chamfer matching in cluttered scenes.

[30] B.R Vatti. A generic solution to polygon clipping. In *Communications of the ACM*, pages 56–63, 1992.

[31] D. Zorin, P. Schroder, and Wim Sweldons. Interpolating subdivision for meshes with arbitrary topology. In *Siggraph 1996*, pages 189–192, 1996.