Vehicle Recognition in Cluttered Environments

A Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By

Gerald Dalley, B.S.

* * * * *

The Ohio State University

2002

Master's Examination Committee:

Dr. Kim L. Boyer, Adviser

Dr. Patrick J. Flynn

Approved by

Adviser
Department of Electrical
Engineering

ABSTRACT

We propose a method for automatic target recognition using an airborne laser range finder for targets hiding under tree canopies. Our system attempts to infer local, low-order, relatively flat surfaces and then reconstruct a mesh model of the object to be identified. We then segment this mesh using a spectral clustering algorithm to find larger shapes that make up the object, and we recognize the target using a partial graph matcher. Given the noise and clutter levels, we demonstrate promising results, with two of our objects achieving 100% recognition rates.

In this thesis we describe the final method we implemented as well as the shortcomings of several alternative approaches attempted. For segmentation, we found that a previous method based on local curvature estimates could not properly account for swift changes in surface orientation. We also explored using a robust surface fitting technique, but found that it produced unreliable segmentations. Finally, we investigated a probabilistic framework for the segmentation algorithm and we explain why that framework fails to work as well for our data as the more heuristic method used.

Dedicated to my loving wife, Dianna.

ACKNOWLEDGMENTS

I would like to recognize the part that each of these has had in my work...

- **Dianna**, my wife, for her constant love and support.
- **My Dad** for instilling in me curiosity and the desire to be an engineer.
- **Ravi Srikantiah** for his thesis and source code that served as a starting point for this work, and for making me wonder who *pam* was.
- **Kanu Julka** for working with me on this project.
- **Rick Campbell** for helping getting me started when I first joined SAMPL.
- The rest of the **SAMP Lab** members with whom I have had the pleasure of working for asking and answering timely questions and making the work here fun.
- **Prof. Flynn** and **Prof. Boyer** for introducing me to the challenging field of computer vision and acting as my advisors.
- **Prof. Dey** for providing the `cocone` software and for his introductory course on computational topology.
- **DAGSI**, the State of Ohio program that funded my work (grants SN-OSU-00-04 and OSURF#739180).

VITA

March 20, 1975	Born - Rochester, MN, USA
1998–2000	**Lucent Technologies.** Co-op. Developed web-based user interfaces and tools for telephony management systems.
2000	**The Ohio State University.** B.S. Electrical and Computer Engineering, with Distinction.
2000-2002	**The Ohio State University.** Graduate Fellow.
Summer 2002	**Microsoft.** Research Intern

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in:

Computer Vision	Prof. Kim Boyer and Prof. Patrick Flynn
Digital Signal Processing	
Computer Engineering	

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Algorithms

CHAPTER 1

INTRODUCTION

1.1 Problem Description and Motivation

Our general objective is to reliably detect and recognize military and civilian vehicles in forested areas given a set of 3D points describing the scene. In particular, we focus on the object recognition portion of this problem.

Automatic recognition of objects from sensed data is a field of study that has attracted a great deal of attention over the years, especially within the computer vision community. In general, it is an under-constrained, ill-posed problem, yet one excellent and readily-available system exists: the human visual system (as well as the visual systems of other animals, to a greater and lesser extent). Unfortunately, this system has its limitations. Years of training are required to develop each system, camouflage on military vehicles can be very effective at foiling the system, and there are increasingly high political costs to placing humans in harm's way. As such, there is an incentive to develop machine-based methods of automatically recognizing objects of interest and being able to differentiate between, say a school bus full of children and a tank. A practical system also should be able to excel where more traditional

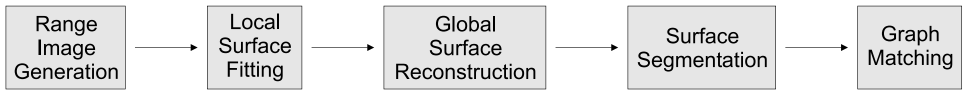


Figure 1.1: **Recognition engine steps.**

visual and photographic techniques fail, such as when vehicles of interest are located under tree canopies, as will be studied in this thesis.

One of the most basic assumptions we have made in designing our system is that vehicles are composed of large, low order, generally flat, piecewise smooth surface segments. In other words, they consist of nearly flat roofs, smoothly-curving hoods, side panels, and other such parts instead of looking like randomly scattered noise. In contrast, natural vegetation exhibits much less structure and organization and comprises more intricate surfaces.

1.2 Organization of the Thesis

In Fig. 1.1, we give an overview of the recognition system. Each of the blocks in the diagram are explained in a chapter of this thesis. Chapter 2 describes how a hypothetical deployed system might generate the set of 3D points describing the scene and how we are synthesizing this process. In Chapter 3, we describe how we estimate the local surface properties of the object of interest given a dense set of points. Chapter 4 describes a method we are using to estimate the surface of the entire object of interest given a sparse set of points. In Chapter 5, we describe how we segment the reconstructed global surface to discover the parts of the object. Chapter 6 describes how we recognize the identity of the object of interest using a graph matching technique based on the segmented parts. Chapter 7 summarizes the contributions of our work and ways in which our work may be enhanced and extended.

In each chapter, we explore relevant prior work that has been used to accomplish the task described in it.

CHAPTER 2

RANGE IMAGE GENERATION

The system described in this thesis is most directly intended to explore issues relating to a future airborne military automatic target recognition system. In our working scenario, an unmanned airplane flies through a geographical region suspected of containing hostile forces in the form of military vehicles hiding in forested areas. Among the sensor equipment on this aircraft is a time-of-flight range sensor that takes range images[1] of places likely to have such enemy vehicles. Where there is evidence of vehicle-like structure in such places, an object recognition system is employed to determine what type of vehicle is present, if any. Pictorially, the goal of our system is to take a scene such as the one rendered in Fig. 2.1a and produce a recognition result such as the one depicted in 2.1b.

Some of the issues involved with such a system include

1. Determination of places likely to contain vehicles in the large geographical region

2. Accurate and fast 3D sensing

3. Alignment (registration) of the 3D data from different range images

[1]A range image gives partial 3D data about a scene. It is effectively a 2D image where the pixel value is a distance measure instead of a reflectance value.

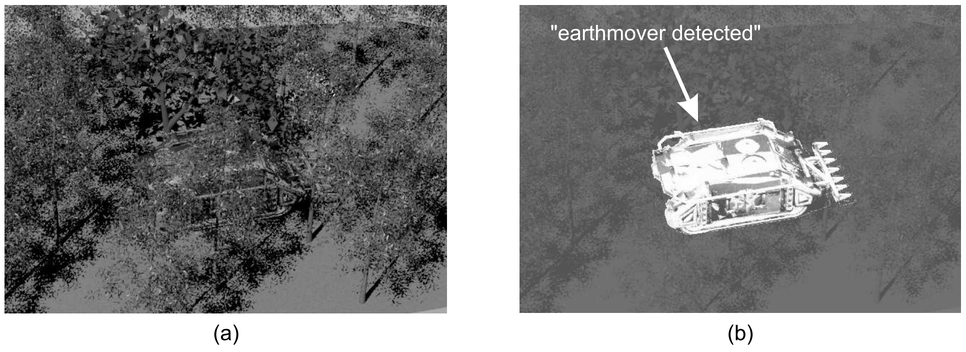


Figure 2.1: **System Goal.** The goal of our system is to take a scene such as the one rendered in (a) and produce a recognition result such as the one depicted in (b).

4. Determination of which parts of the range images correspond to a vehicle vs. vegetation (initial detection)

5. Object recognition given the vehicle range points

In this thesis, we will focus primarily on the final issue: that of object recognition. Because we do not expect the initial detection to be perfect, we also allow some spurious 3D points to be sent through the system. The other issues just described are outside the scope of this work.

Because the hypothetical sensor just described does not yet exist, we are simulating it by producing the range images using a ray tracing program. When ray tracing, we wish to find the color and/or depth value for each point (pixel) on a regular 2D grid called the screen, as shown in Fig. 2.2. For our system we use an orthographic camera because we envision the sensor being used at a high stand-off distance with a narrow view angle. In this case when ray tracing, there is a sensor for each pixels position on the screen. A ray is passed from the sensor through the screen and continues until it intersects the first object (the shaded pentagon). If we are producing

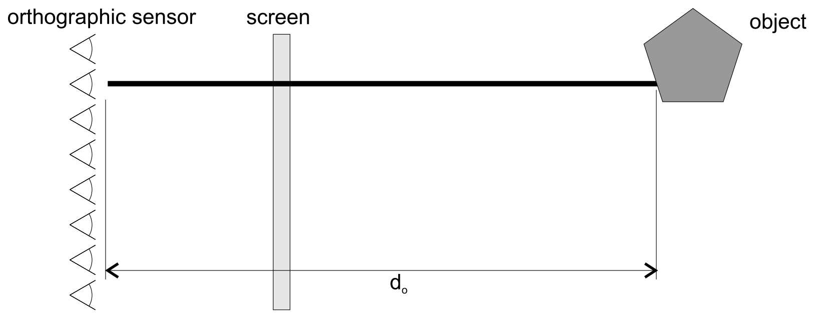


Figure 2.2: **2D Ray Tracing.** Ray tracing assuming an orthographic camera is being used.

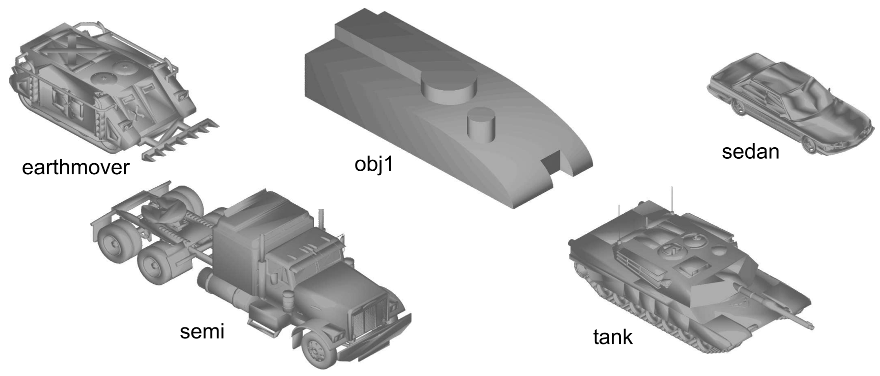


Figure 2.3: **Renderings of the Objects Used.** The five objects we used are shown as intensity image renderings. All five are shown to the same scale.

an intensity image, we calculate the reflectance values from any light sources we have defined (not shown). When producing range images, d_o is stored. For more detail on ray tracing techniques, the reader may refer to virtually any introductory graphics programming text including the one by Foley *et al.* [13] or by Watt and Watt [31].

To generate our data, we have modified a version of the popular freeware POV-Ray ray tracer [39] to output the 3D global coordinates of each range point. We have selected CAD models of five objects to test our recognition engine: *earthmover*,

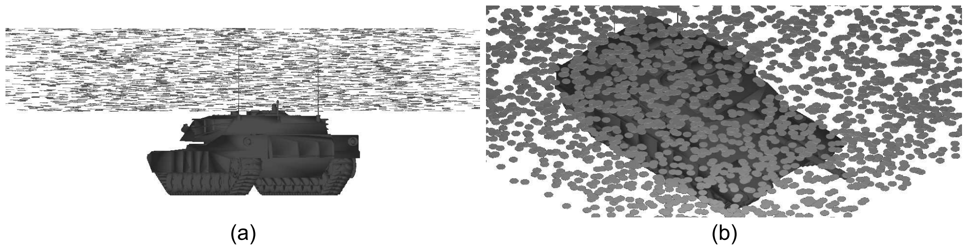


Figure 2.4: **Foliage.** Two views of the tank object occluded by our pessimistic simplified foliage model.

obj1, *sedan*, *semi*, and *tank*. Intensity image renderings of each are shown in Fig. 2.3. For reference purposes, *obj1* is $3.1 \times 8.1 \times 2.0m$ in size. All range images taken are 512×512 pixels, yielding an approximate pixel-to-pixel distance of $30mm$ using an orthographic projection.

We created three scenes which we have called *circle*, *flyby*, and *unoccluded*. The first two have simulated vegetation and the last has only the object and a ground-plane. For the *circle* scene, a circular flight path was taken around the object of interest with an angle of elevation of 75^o with 12 evenly-spaced images taken about the circle. For the *flyby* scene, the camera is flown in front of and behind the object in two passes, taking a total of 20 range images. The camera begins taking range images when it is placed at 45^0 to the object's long axis in the ground plane. As was the case with the *circle* scene, the angle of elevation of the camera is 75^o. For the *unoccluded*, we place the camera at each vertex of a dodecahedron centered on the object and retain only those range images produced by cameras above ground level (10 of them).

For the foliage, we initially used tree models designed to look realistic in intensity images such as those used to generate Fig. 2.1a. We unfortunately found that these models were both extremely time consuming for the renderer and did not provide much occlusion for the vehicles when all of the range images in a scene were put together. After attempting a few other foliage models, we settled on a rather simple one that was easy to render yet provided sufficient occlusion. For each scene, we created 500 randomly placed thin discs. These discs were placed in a vertical band starting at the top of the vehicle and going up 2 meters. They were spread out over a square 12.2×12.2 meters and had a radius of $100mm$. Fig. 2.4 shows two example renderings of *tank* using this foliage model.

After generating the scenes, we added isotropic Gaussian noise to the range points where the standard deviation of the noise was 0, 2, 4, 8, 16, and $32mm$.

We assumed that the initial detection system will be able to localize the object of interest reasonably well, so we retained all points that lay within a bounding box centered on the object $10m$ long, $4m$ wide, and $4m$ tall.

In summary, we generate a set of synthetic range images for each object pictured in three scenes. The *unoccluded* scene has no vegetation, has the camera positions placed even about the view sphere, and is used to build the modelbase. The *circle* and *flyby* scenes include pessimistically-modelled vegetation and use plausible aircraft flight paths. In all cases, varying levels of isotropic Gaussian noise are added to the range point coordinates.

CHAPTER 3

LOCAL SURFACE FITTING

In the introduction (Section 1.1), we stated one of our basic assumptions: vehicles are well-described by large, piecewise smooth, "low order", flat surface segments. Fig. 3.1 shows some example curves that represent these concepts assuming that we consider each side of the roughly rectangular shapes to be different "pieces." We assume that vehicle pieces in which we are interested look like (c) and (d), not like (a) or (b).

Our recognition system is designed to match groupings of points that imply these well-behaved surfaces. In order to do so, we must first form reliable local surface estimates.

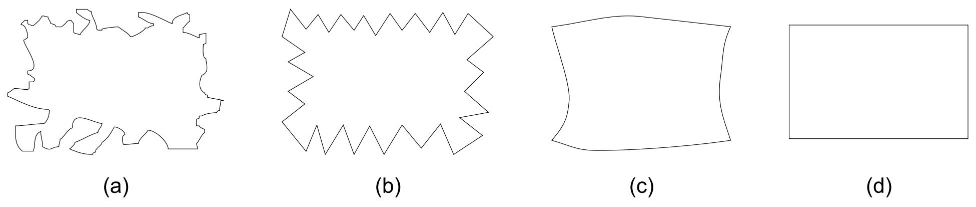


Figure 3.1: **Curves that do and do not consist of large, piecewise smooth, flat, low order segments.** Various types of curves that each have four "flat" pieces roughly corresponding to the sides of a rectangle. (a) non-piecewise smooth and high order (b) low order, but non-piecewise smooth (c) and (d) piecewise smooth and low order.

We have found that our surface segmentation algorithm (Chapter 5) is sensitive to the manner in which the local surfaces are estimated. First, the segmenter does not deal well with large numbers of range points (*e.g.* our *tank* object has over 220,000 raw points). Secondly, the surface fitting process tends to produce smoothed surface estimates if not done properly. Our segmentation algorithm relies on discontinuities between the low order surface segments, which smoothing destroys.

In Section 3.2, we will describe how we subsample the initial point cloud (collection of unorganized points) and do so in a way that avoids the smoothing problem just mentioned. We describe the particulars of our local surface fitting using rotated biquadratics in Sections 3.3 and 3.4.

3.1 Prior Work

Before describing the methods we have used for local surface description, we give a brief overview of common local surface descriptions used by researchers to compactly describe surfaces for object modelling and recognition.

By far, the most common method for representing surfaces is as piecewise planar shapes, or polygonal meshes. This representation is the simplest and is very amenable to hardware implementations for rendering purposes. For objects that are indeed planar, the approximation is very good. Unfortunately, this representation has some major drawbacks. It may poorly represent shapes having many bumps, wiggles, and tight curves unless an extremely large number of polygons are used. Moreover, it is not always descriptive enough to accurately convey information needed for surface analysis.

One of the more common types of analysis desired for recognition purposes is the calculation of surface curvature. Flynn and Jain [11] have explored a number of different techniques for estimating curvature on meshes, including using a bicubic polynomial fit as opposed to only using the planar polygonal facets. We have adopted Srikantiah's usage of a slightly simpler biquadratic [26]. This method will be described in more detail in sections 3.3 and 3.4

3.2 Point Selection

Before actually estimating the surfaces, we must decide where we need the estimates so that we avoid overburdening the recognition system. We have experimented with several methods, and in the remainder of this section, we describe the two most interesting methods of these.

In our discussion, we use the term "point cloud" to refer to all of the original range points. The "subsampled points" are those that have been selected to have surface estimates made and are preserved in the recognition pipeline after the local surface estimation step.

3.2.1 Random Selection

First, we tried randomly selecting 5% to 10% of the points from the point cloud and estimating the local surface characteristics by looking at their local neighborhoods. We used this approach because we wanted to avoid artificially introducing structure in our data by evenly spacing the points. Once the points were selected, we estimated the surface by performing a least-squares biquadratic fit to all *subsampled points* within some radius.

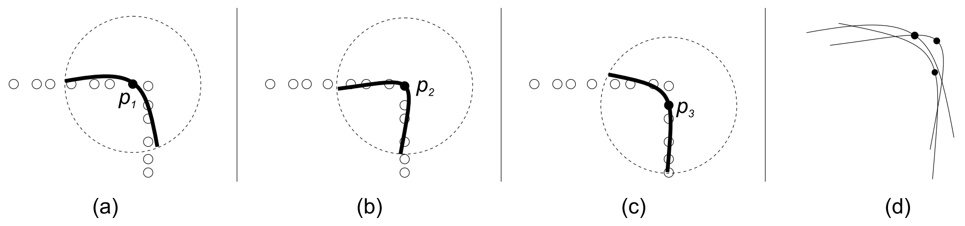


Figure 3.2: **Quadratic fits near a corner.** (a), (b), and (c) show approximate curve fits given the local neighborhood defined by the filled point and the dashed circle. (d) shows the three superimposed curve fits, zoomed in.

As a baseline, we chose a neighborhood radius such that about 90% of the points had local neighborhoods with at least 8-10 other points. This resulted in very tight, very local biquadratic fits. Unfortunately, when we began trying to group points based on local fits (chapter 5), we saw very poor results. We experimented with increasing the neighborhood radius by a factor of eight and we still could not get good segmentations. Despite attempts to adjust the neighborhood radius and various parameters of the segmentation algorithm, we typically had one or more of the following problems: execution times on the order of hours or days, non-convergence of the segmentation algorithm, and meaningless segmentations. In summary, this approach was unstable.

Although we found that additional factors were at play, a fundamental problem with the approach just given was that we had introduced smoothing into the surface fitting procedure. We will illustrate the problem using a 2D analogy. In Fig. 3.2a-c, we see three local curve fits near a corner, given three neighboring center points. As seen in Fig. 3.2d, within the local neighborhoods, these curve fits are quite similar, even though points 1 and 3 arose from different smooth curves, and point 2 from a boundary between these curves. If noise were added to the point locations, the

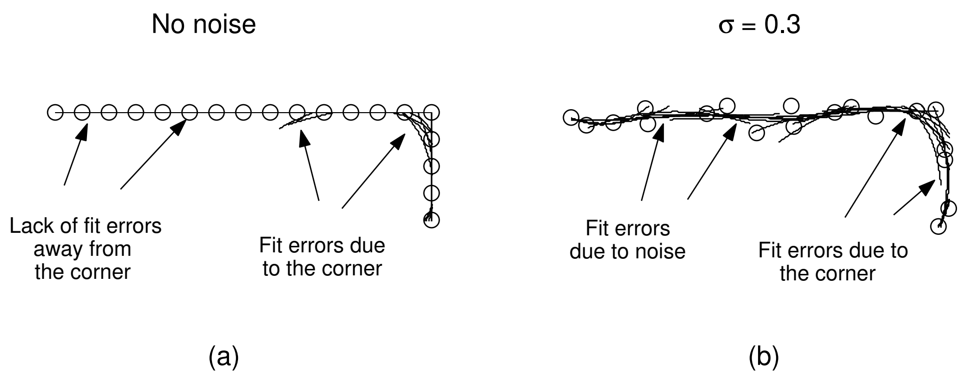

No noise
σ = 0.3
Lack of fit errors away from the corner
Fit errors due to the corner
Fit errors due to noise
Fit errors due to the corner
(a)
(b)

Figure 3.3: **Noise effects on quadratic fits.** (a) Shows a series of quadratic fits for a neighborhood radius of 3 with no noise. The only fit errors present are due to the corner. (b) Shows the same points, with additive isotropic Gaussian noise ($\sigma = 0.3$). The fit errors far from the corner are very similar to those near the corner.

differences between these fits would look very similar to the differences between fits that are only estimated from points belonging to the same curve (see Fig. 3.3). In contrast to this situation, we would like all three points to produce very different surface fits even though they are closest neighbors.

3.2.2 Voxelized Selection

To avoid smoothing across surface parameter discontinuities as just seen, we would prefer that each range image point participate in one and only one local surface fit. Put more precisely, we would like the following to be true

$$\bigcup_i N_i = P \qquad \bigwedge \qquad \bigcap_i N_i = \oslash \tag{3.1}$$

where P is the set of all range points, N_i is the set of points in the local neighborhood of $s_i \in S$, and S (not shown in Eqn. 3.1) is the set of points selected by subsampling.

Two possible methods of creating the disjoint neighborhoods described in Eqn. 3.1 are as follows. First, we could continue to randomly select S, as was done in section 3.2.1, but restrict each point in P to the neighborhood of its closest point in S. A second approach is to partition the space into voxel (cube-shaped) bins. All points in voxel i are assigned to neighborhood N_i, and the point in N_i nearest the center becomes s_i, the point retained after subsampling. Because of its computational efficiency, we have chosen the second approach.

Once we have assigned each input range point to a voxel, we evaluate each voxel to determine whether it appears to represent a surface of interest. If a voxel has too few points, then we cannot reliably estimate the local surface attributes or there may effectively be no surface at all, so we discard it. Additionally, if we have a high fit error (Section 3.4), then the voxel does not contain a surface well described by our model (*e.g.* it may contain points from scattered leaves), and again we discard it. For our baseline tests with noiseless data, the voxels are 128mm on edge, we require each one to have at least 32 points, and we require the root mean square (RMS) fit error to be 8mm or less.

3.3 Local Coordinate Frame

Once we have selected a set of points, we are ready to estimate the local surface properties. For the surface fitting that will be described in the next Section, we need to establish a local coordinate system for the point neighborhood that places the surface normal in the direction of a preset axis, in our case the w-axis. For this, we use a standard principal components analysis (PCA) algorithm that works by finding the eigenvalues of the sample covariance matrix:

$$\mathbf{S}_i = \frac{1}{|N_i|} \sum_{j=1}^{|N_i|} \left[n_{i_j} - \hat{n}_i\right] \left[n_{i_j} - \hat{n}_i\right]^T \tag{3.2}$$

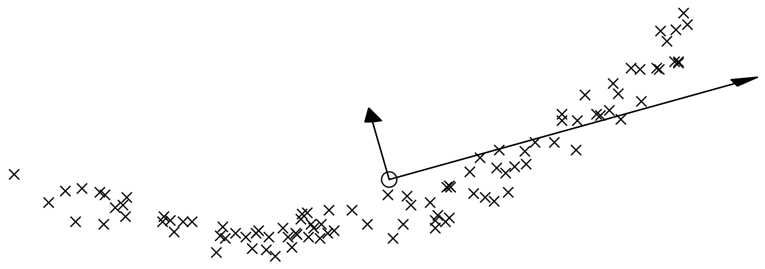


Figure 3.4: **PCA of a curve.** The local coordinate system for a set of 2D points was found via principal component analysis (PCA). X's are the input points, O is the centroid of the X's, and the two arrows point in the directions of the two eigenvectors of the covariance matrix of these points and are scaled according to the square root of their eigenvalues.

where n_{i_j} is the j^{th} point in the neighborhood, $|N_i|$ is the number of elements in neighborhood i, and $\hat{n}_i$ is the centroid $\hat{n}_i = \sum_j n_{i_j}/|N_i|$.

Eigenvector v_1 of $\mathbf{S}$ corresponding to the smallest eigenvalue points in the direction of least spread of the points in N_i. Likewise, eigenvector v_3 corresponding to the largest eigenvector points in the direction of greatest spread of the points. These eigenvectors are guaranteed to point in orthogonal directions. Fig. 3.4 shows an example of the results of the PCA analysis for a set of 2D points. A more detailed discussion of the theoretical underpinnings of PCA can be found in Strang's linear algebra book [28] by combining the discussion of covariance matrices (pp. 203–205) with the sections explaining Rayleigh-Ritz (pp. 347–360). Additionally, Flynn and Jain present justifications for using this approach [10].

Once we establish this new coordinate system, we create a homogeneous transformation matrix that converts from (x, y, z) coordinates to (u, v, w). Homogeneous transforms are represented by 4×4 matrices and are capable of combining both rotation and translation into a single matrix multiplication operation. The homogeneous system combines the usual (x, y, z) coordinates with an α component that can be thought of as a scale parameter, *e.g.* $[\tilde{x}, \tilde{y}, \tilde{z}, \alpha]^T$ refers to a point at $[\frac{\tilde{x}}{\alpha}, \frac{\tilde{y}}{\alpha}, \frac{\tilde{z}}{\alpha}]^T$ in the usual Cartesian coordinate system. The transform we use is:

$$p_{uvw\alpha} = \mathbf{T} p_{xyz1} = \begin{bmatrix} \cdots & v_3 & \cdots & -\hat{n}_i.x \\ \cdots & v_2 & \cdots & -\hat{n}_i.y \\ \cdots & v_1 & \cdots & -\hat{n}_i.z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{xyz1}.x \\ p_{xyz1}.y \\ p_{xyz1}.z \\ 1 \end{bmatrix} \tag{3.3}$$

where $\hat{n}_i$ is the neighborhood centroid and $\triangle.x$ is the x component of vector $\triangle$.

This transformation moves p into a coordinate system centered at n_i and where the u direction is aligned with the the largest eigenvector, v_3, the v direction with v_2, and the w direction with the smallest eigenvector, v_1. For a fuller discussion of homogeneous coordinates and how they can be used to combine both rotation and translation into a single matrix, please refer to the OpenGL Programming Guide [34].

3.4 Biquadratic Surface Fit

Once we have transformed our neighborhood of points into its local (u, v, w) coordinate system, we are ready to fit a surface to those points. We experimented with doing planar fits (*e.g.* only using the PCA output), but found that planes were not descriptive enough to describe our object parts well. Instead, we adopted Srikantiah's least squares biquadratic fit [26, pp. 28–29]. For it, we wish to describe the local surface as the following function:

$$w = f(u, v) = a_0 u^2 + a_1 uv + a_2 v^2 + a_3 u + a_4 v + a_5 \tag{3.4}$$

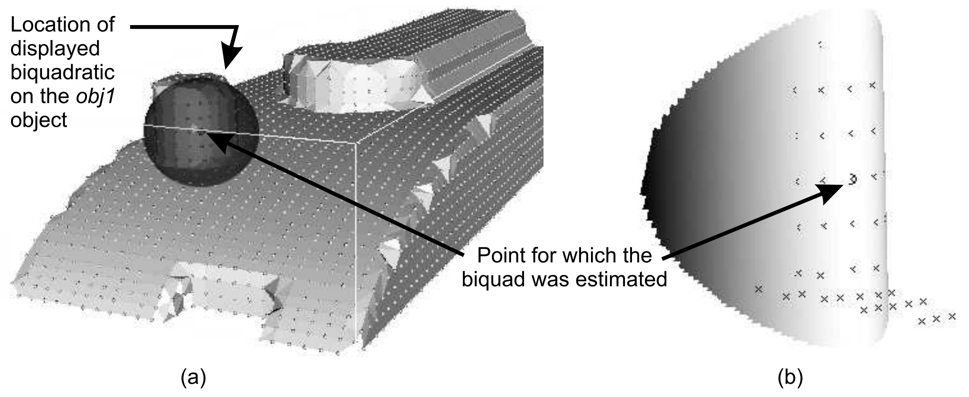

Location of displayed biquadratic on the *obj1* object
Point for which the biquad was estimated
(a)
(b)

Figure 3.5: **Biquadratic fit on *obj1*.** Sample biquadratic fit extended out on the global surface. (a) shows the whole object. (b) shows a point (circled) whose voxel was used to estimate a biquadratic, points from neighboring voxel centroids (x's), and the biquadratic (shaded surface). The biquadratic surface in (b) has been clipped by the highlighted sphere shown in (a).

For the collection of neighborhood points, we can express this equation in matrix form as follows:

$$\mathbf{B}a = \begin{bmatrix} u_0^2 & u_0v_0 & v_0^2 & u_0 & v_0 & 1 \\ u_1^2 & u_1v_1 & v_1^2 & u_1 & v_1 & 1 \\ u_2^2 & u_2v_2 & v_2^2 & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \approx \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \tag{3.5}$$

By taking the Moore-Penrose pseudo-inverse of **B**, we are able to obtain a least-squares solution to this problem that minimizes the squared distance between each point and the surface in the w direction.

Note that in order for **B** to be full-rank there must be at least six points in the neighborhood, selected to produce linearly independent rows (non-collinear, for example). More points are necessary for accurate surface estimation as noise is added to the system. Fig. 3.5 shows an example of a local biquadratic fit extended out beyond the original voxel.

3.5 Summary

In summary, our surface segmentation scheme (to be described in Chapter 5) requires that we have estimates of the local surface characteristics for a subset of the original points. To avoid excessive smoothing, we partition the original region of interest into cubic voxel bins. Out of each of these bins, we retain at most one point and the information derived from fitting an oriented biquadratic surface to the points in the bin.

In the rest of this thesis neighborhoods, additional surface fits, and meshes all are based on the subsampled points only, unless explicitly stated otherwise. The original range points discarded by subsampling are not examined further.

CHAPTER 4

GLOBAL SURFACE RECONSTRUCTION

After completing the local surface fits described in the previous chapter, we are not yet prepared to begin grouping points into larger surfaces. This is because our points have no topology information; all we have are a set of attributed points scattered in space with no encoded connectivity. In other words, our data look like Fig. 4.1a, without the globally connected surface of 4.1b.

This topological information is useful for a number of reasons:

- It allows for easy, unambiguous identification of nearest neighbors.

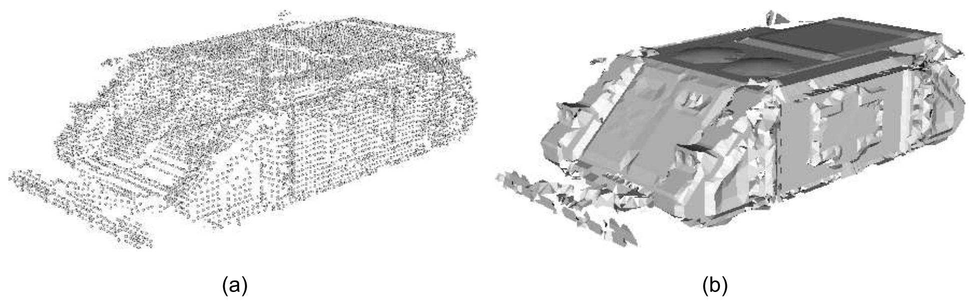


Figure 4.1: **The *earthmover* object without and with topology.** (a) with only its retained point samples and (b) after global surface reconstruction (*e.g.* with topological information). Voxel size used: 64mm.

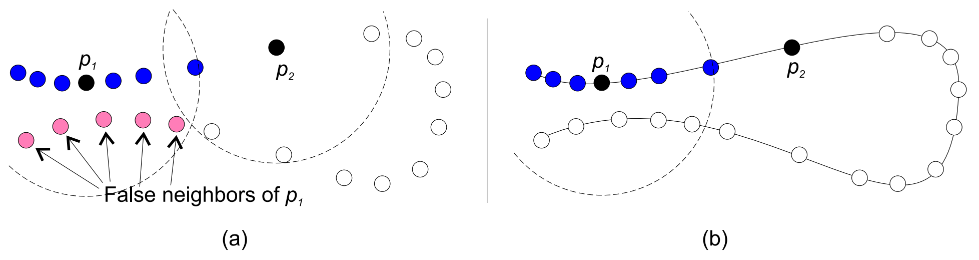


Figure 4.2: **False neighbors due to lack of topology.** (a) some points are incorrectly grouped with point p_1. (b) Introducing topological information alleviates this condition.

- Searches over sets of small cardinality can be implemented with efficient and simple recursive algorithms.
- Renderings such as Fig. 4.1b become possible and ease the qualitative evaluation of results.
- It allows us to easily avoid incorrect grouping of nearby surfaces.

To explain the “incorrect grouping” problem, consider the set of points in Fig. 4.2. Suppose that we wish to group points that fall within some distance of each point. For point p_2 to be grouped with either of its immediate neighbors, we must make this distance relatively large. If we use this same radius for point p_1, many points that obviously[2] come from a separate surface get included in its neighborhood. In Fig. 4.2a, we would like only the darkly shaded points near p_1 to be used, not the lightly shaded (red) points. If we reconstruct the surface as in Fig. 4.2b and walk along our mesh, we can avoid considering the “false neighbors.”

For these reasons, we perform surface reconstruction on the points selected for receiving local surface fits (Chapter 3). Note that we did not reconstruct the surface

[2]It is obvious to the human observer who can observe the overall point configuration

before doing the local fits because the reconstruction is a computationally expensive process when the full set of range points are used. Additionally, at the biquadratic estimation stage, intra-voxel topology is a non-issue because the voxels are small and we are only interested in voxels representing surface patches which have low curvatures relative to the voxel size.

In Section 4.1 we describe existing methods for performing surface reconstruction. We then explain some preliminary concepts in Section 4.2 that are necessary to understand a 2D curve reconstruction algorithm (Section 4.3) and its generalization to 3D surfaces (Section 4.4) that we use. In Section 4.5 we describe some mesh post-processing that we perform to clean up our reconstructed surface.

4.1 Prior Work

A number of different techniques have been proposed for reconstructing surfaces from unorganized point clouds. One popular method proposed by Hoppe *et. al* [16] and adapted by Curless and Levoy [6] finds zero locations for the signed distance function. The surface is then reconstructed using an algorithm such as Marching Cubes [18]. Though intuitive, this method makes no guarantees about the topological correctness of the reconstructed shape.

In order to add these guarantees, Amenta *et. al* developed the *crust* algorithm which is based on analysis of the Voronoi diagram of the sample points [1]. `Cocone`, a descendant of *crust* is based on the same concepts, but is faster and has weaker sampling constraints [2]. We employ `cocone` for our surface reconstruction and explain the algorithm here.

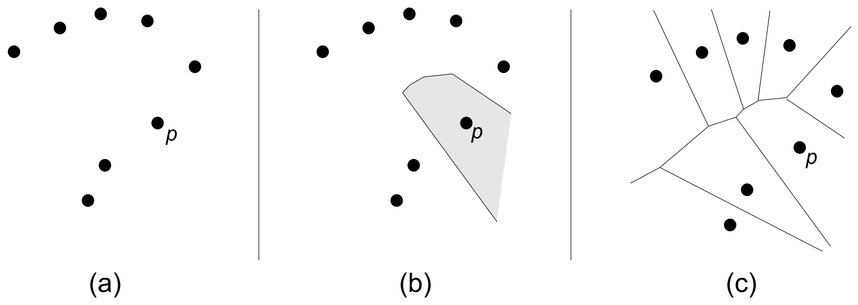


Figure 4.3: **Voronoi diagram of a sampled curve.** (a) a set of points sampled from a curve (b) the Voronoi cell of point p–shaded (c) the Voronoi diagram of the entire point set

4.2 Preliminaries

In discussing `cocone`, we will begin by exploring some preliminary concepts from computational geometry and topology. After presenting a simple curve reconstruction technique, we will describe the `cocone` surface reconstruction algorithm itself.

4.2.1 Voronoi Diagrams

The first preliminary topic we will discuss is the Voronoi diagram of a set of sample points. To simplify this discussion, we restrict ourselves to shapes and point sets embedded in 2- or 3- dimensional spaces.

Consider a point p from a set of sample points, P, such as the one shown in Fig. 4.3a. The *Voronoi cell* of p is the set of all points in the space closer to p than to any other sample points in P (note that a "sample point" is a location that has been measured or identified, whereas a general "point" is any location in the space). In Fig. 4.3b, the shaded region is the Voronoi cell of p. The *Voronoi facets*, *edges*, and *vertices* are the facets (for 3D only), edges and vertices of those cells. The *Voronoi*

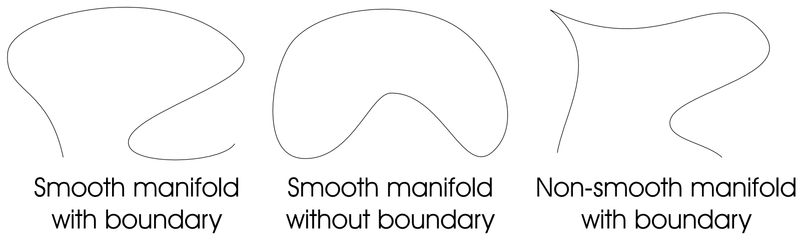


Figure 4.4: **Three examples of 1D manifolds embedded in a 2D space.**

diagram is the set of all Voronoi cells, facets (for 3D only), edges, and vertices for all of the sample points being considered (Fig. 4.3c). Note that the Voronoi facets and edges are always perpendicular to the line segment connecting the two sample points in cells separated by the facet or edge.

4.2.2 Medial Axis

For the Voronoi diagram, we assumed a measured set of points drawn from a shape. For the medial axis, we will consider not the sample points but the original shape itself. To be precise in our definition of "shape," we will consider N-1 or lower dimensional smooth manifolds (possibly) with boundaries embedded in N dimensional spaces. For example, in a three-dimensional space, a "shape" is a (possibly non-closed) surface, curve, or lone point without creases, folds, or hard edges. Fig. 4.4 shows some examples of 1D manifolds (curves) embedded in a 2D space.

A *medial ball* is a ball (*i.e.* a circle in 2D, a sphere in 3D, *etc.*) that touches the shape tangentially on at least two points. For a manifold without boundary, the *medial axis* is defined to be the set of points equidistant from at least two surface points, or as the locus of all of medial ball centers. The precise definition of the medial axis for manifolds with boundaries is beyond the scope of this paper. Fig. 4.5

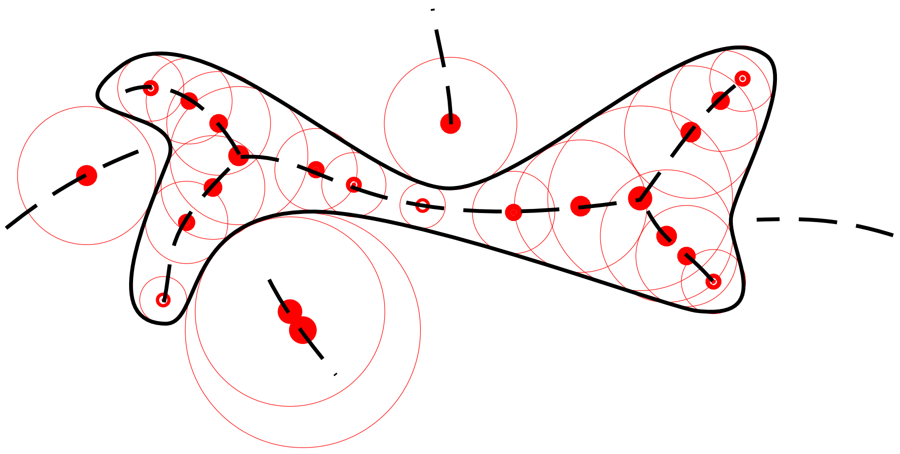


Figure 4.5: **Medial axis of a closed curve.** A shape (solid line) with several medial balls (circles) and its medial axis (dashed lines)

shows an example of a 1D shape (solid line) and several of its medial balls (circles) whose centers form the medial axis (dashed lines).

4.2.3 ϵ-sampling

The distance to the closest point on the medial axis from a given point on the shape is called the *feature distance*. For the curve and surface reconstruction algorithms we will discuss, it is important to sample the shape in a manner proportional to the local feature distance. Conceptually, regions of high curvature are close to the medial axis and must be sampled more finely to capture all of the surface detail. Flat, well-separated regions are far from the medial axis and only a few sample points are necessary for properly reconstructing those portions of the shape. A shape that has been ϵ-*sampled* is one that, for each sample point, there is a neighboring sample point at most ϵ times the feature distance away ($0 < \epsilon < 1$ in general). Fig. 4.6 shows a portion of the curve from Fig. 4.5 that has been ϵ-sampled where $\epsilon = 0.5$.

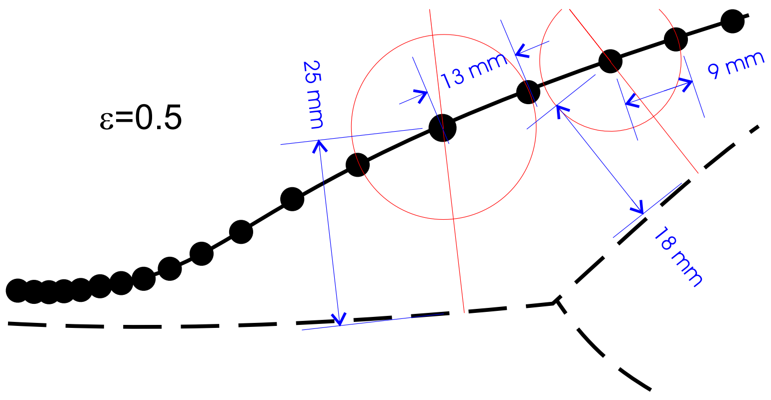


Figure 4.6: $\epsilon-$**sampling.** A portion of a curve (solid line) with its medial axis (dashed line) that is ϵ-sampled where $\epsilon = 0.5$.

4.2.4 Correctness

The curve and surface reconstruction algorithms about to be discussed are proved to be topologically and geometrically "correct" when $\epsilon \leq 1/3$ [8] and $\epsilon \leq 0.06$ [2], respectively. In lay terms, topological correctness implies that the reconstructed shape has the same number of holes and pieces as the original shape. Geometrical correctness in this case implies the following for surface reconstruction when $\epsilon \leq 0.06$ [2], given the feature distance, $f(p)$ for each sample point p:

- Any point on the reconstructed mesh is at most $0.088f(p)$ from the true surface.
- The maximum error in any triangle face normal is 42°.
- The maximum error in any vertex normal is 28°.

In practice, ϵ may be much larger than the values referenced here and still yield a correct or nearly-correct reconstruction. In our experience, the algorithm generally fails gracefully when undersampling conditions occur (as they frequently do for us).

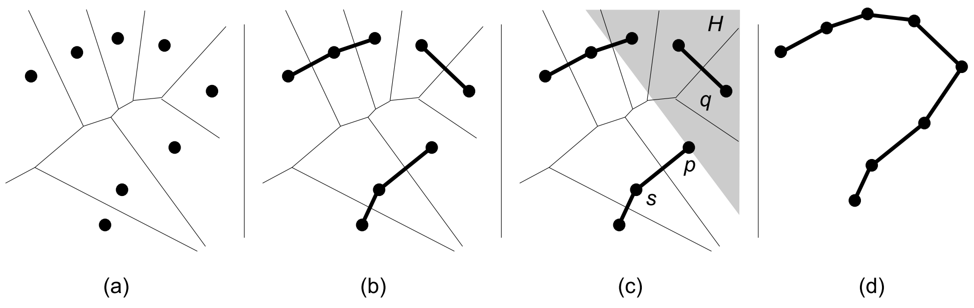


Figure 4.7: **Curve reconstruction.** (a) The Voronoi diagram of a set of points sampled from a curve. (b) All nearest neighbor pairs are connected. (c) Point p with the half-plane, H opposite its nearest neighbor, s shaded. Point s is the closest to p in H. (d) The final reconstructed curve.

4.3 Curve Reconstruction

Algorithm 1 Curve Reconstruction

```
Compute the Voronoi diagram of the sample points
for all points $p$ do
  Find the nearest neighbor $s$
  Connect $p$ and $s$ with a line segment $ps$
  Form the half-plane $H$ whose edge runs through $p$ and is perpendicular
      to $ps$ and does not include $s$
  Find the nearest neighbor $q$ in $H$
  If segment $pq$ does not yet exist, connect $p$ and $q$ with a line segment $pq$
end for
```

Fortunately, reconstructing a curve is intuitive and straightforward, as given in Algorithm 1 and illustrated in Fig. 4.7. First, the Voronoi diagram of the point set is computed (Fig. 4.7a). Then, line segments are created connecting each point to its nearest neighbor, as in Fig. 4.7b. For each point that does not have two edges connected to it, a half-plane is temporarily constructed. This half-plane's edge is perpendicular to the existing line segment (connecting the current point to its nearest

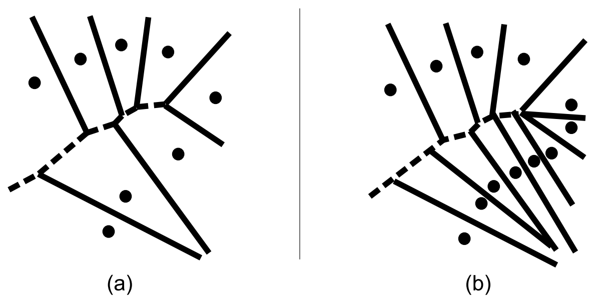


Figure 4.8: **Elongated vs. wide Voronoi cells.** Solid lines denote Voronoi edges caused by close neighboring points. Dashed lines denote Voronoi edges close to the medial axis. (b) has been sampled more densely and has more elongated cells.

neighbor) and runs through the point (Fig. 4.7c). Next, the nearest neighbor to the point in the half-plane is used to construct the point's second line segment. If there are no points in the half-plane, then an endpoint has been detected for the curve. Once this procedure has been completed for all points, the curve is reconstructed (Fig. 4.7d). As stated before, assuming that the curve is $\epsilon-$sampled at $\epsilon \leq 1/3$, this algorithm is provably correct.

4.4 Cocone Surface Reconstruction Algorithm

To reconstruct surfaces (2D manifolds with edges embedded in a 3D Euclidean vector space), we need a slightly more sophisticated algorithm that examines the Voronoi diagram of the sample points in a little more detail.

First, we note that Voronoi boundaries (the faces and edges bordering Voronoi cells) are caused by either two neighboring samples being close to each other or by the medial axis being close to a sample point, as shown in Fig. 4.8. The location of the medial axis does not change under denser sampling conditions. As such, when we sample more finely, the Voronoi cells become elongated.

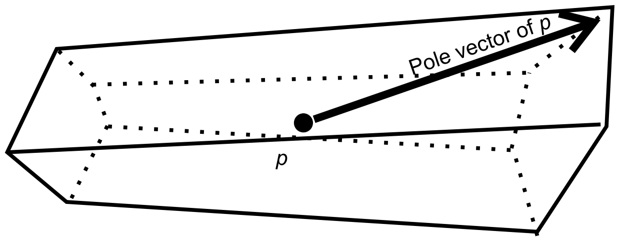


Figure 4.9: **Pole vector in a 3D Voronoi cell.**

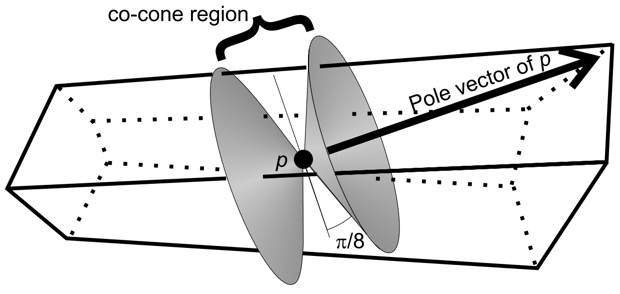


Figure 4.10: **Cocone region.** Co-cone region for the cell from Fig. 4.9

We define the *pole vector* to be the vector going from the sample point to the farthest corner of the Voronoi cell (Fig. 4.9). When the Voronoi cell is unbounded, any vector that does not intersect a cell boundary may be chosen as the pole vector. As the cells get skinnier, the pole vector more closely approximates the surface normal at the sample point. As shown by Amenta [2], when $\epsilon < 0.1$, the pole vector will be within $\frac{\pi}{8}$ radians of the true surface normal.

Once we have found the pole vectors, we define the co-cone region of a Voronoi cell to be the volume between the cones $\frac{\pi}{8}$ above and below the plane perpendicular to the pole vector (Fig. 4.10). The true surface is guaranteed to lie entirely in this co-cone region. With these definitions, we are ready to specify the `cocone` algorithm given in Algorithm 2.

Algorithm 2 Surface Reconstruction

```
Find the Voronoi diagram of the sample points
for all cells in the diagram do
    Find the co-cone region
    For each Voronoi facet in the co-cone region, create a surface edge starting at
    that the cell's sample point and going to the neighboring cell's sample point.
end for
```

After completing this procedure, some cleanup work is necessary to remove false and redundant surfaces and to detect regions of undersampling. These procedures are detailed in [7].

4.5 Mesh Post-Processing

In addition to the cleanup work done by `cocone` itself, we have found it useful to prune out parts of the mesh that tend to confuse the surface segmentation algorithm described in Chapter 5. We prune facets that have fewer than three sides, have large areas, and/or have high aspect ratios. We also prune vertices that have low elevations and/or are isolated points. Facets with fewer than three sides can appear when we prune some but not all of the vertices of a facet.

We first remove any vertices that have too low elevations because points that are near the ground are much more likely to arise from grass, bushes, and other low-level foliage. We are currently removing any points lower than 300mm from the ground. After doing so, we remove any degenerate facets that have fewer than three vertices. Although our analysis algorithms could actually handle these degeneracies, it is useful

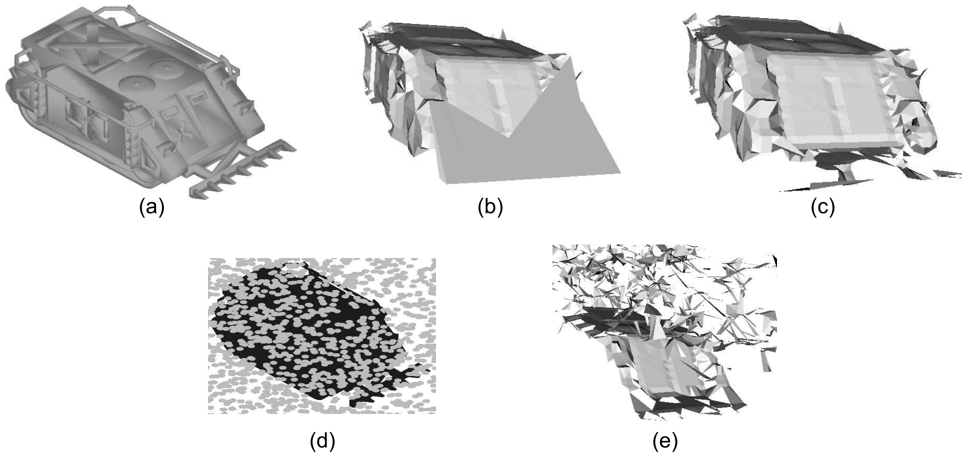


Figure 4.11: **Mesh Pruning Examples.** (a) is a POV-Ray shaded rendering of the *earthmover* object. (b) shows the initial reconstructed mesh, with several false and very large triangles. (c) The large triangles have been removed. (d) A false-color POV-Ray rendering of an *earthmover* scene with simplified-leaf occlusions. (e) The initial mesh. Many false triangles with high aspect ratios have been created that join randomly scattered leaves.

to prune them to ease the rendering of results and these degenerate points quite often lie on what will be future segmentation boundaries.

Because we very frequently violate the required ϵ-sampling conditions, `cocone` occasionally produces false facets such as the ones shown in Fig. 4.11b going from the rake to the top of the vehicle (see Fig. 4.11a for a rendering of the original vehicle shape). We have found that by pruning facets whose area is $40,000mm^2$ or greater that we are able to very reliably remove such false facets yet retain all of the true facets (see Fig. 4.11c).

Our next problem arises because we are subsampling our original point cloud. When we add foliage such as the leaves shown in Fig. 4.11d and then subsample the point cloud, the region that contains the leaves essentially becomes a volume with randomly-placed points scattered in it. The version of `cocone` we are using is designed to extract only surfaces and not volumes[3]. As a result, it tends to create triangles adjoining different leaves, as shown in 4.11e. These false surfaces tend to confuse the surface segmentation and graph matching algorithms. Fortunately, unlike the well-sampled portions of the vehicle surface, the triangles in the foliage area tend to have high aspect ratios (longest edge length / shortest edge). We remove any triangles whose aspect ratio is above 1.75. This process occasionally results in the removal of correct triangles, but not so many that serious problems result.

Once we have removed the spurious triangles, we often are left with orphaned points that belong to no valid triangle. These points are now discarded.

[3]`cocone` has been generalized to handle N-dimensional manifolds with boundaries [9]

CHAPTER 5

SURFACE SEGMENTATION

Once we have a reconstructed surface (Chapter 4) whose points are attributed with local surface estimates (Chapter 3), we are ready to segment the surface into the parts that imply flat, low order surface patches. Once we have these patches or segments, we can attempt to match them to ones in the modelbase in order to identify the object seen.

5.1 Assumptions and Motivation

There is a multitude of recognition algorithms that do not require segmentation to be performed, some of which are mentioned in Section 6.1; however, we have chosen to take a segmentation approach because of several assumptions that we make about how point data will be gathered in an eventual deployed application.

In such a system, we envision a frequent condition where entire vehicle faces are not sampled. This could happen because a vehicle is parked near a large occluding object such as a cliff, another vehicle, a building, or very dense foliage. Additionally, the flying sensor may not have the opportunity to sample the vehicle from enough different viewpoints to see all of the faces. By segmenting the surface, we can apply our recognition techniques to only the parts that were seen.

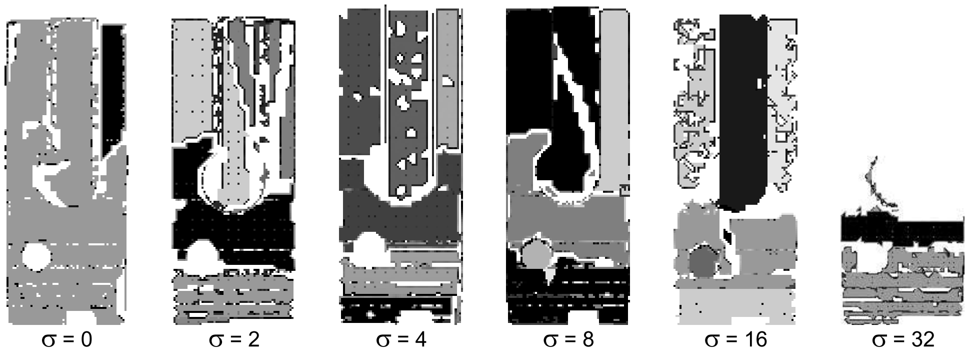


Figure 5.1: **Unreliable segmentations using robust sequential estimators.** These six segmentations are of the same *obj1* object without occlusion given different additive Gaussian noise parameters. Note that segmentation boundaries change significantly with different noise inputs.

Secondly, we do not assume that optimal sampling conditions will be present. A body of techniques look for features such as crease edges, which are very useful features for the human visual system. Unfortunately, we do not expect those edges to be sampled well. The segmentation approach we use places the focus on finding consistent surface estimates and thus takes the focus away from these edges.

We expect a segmentation scheme to be "reliable" in order to be useful for us. By this we mean that the algorithm should be able to provide the same segments under different sampling and noise conditions.

5.2 Prior Work

After briefly discussing some alternate techniques, we will spend most of this section talking about spectral clustering methods for segmentation.

5.2.1 Miscellaneous Segmentation Techniques

Mirza and Boyer [19] used a robust weighted least-squares fit to estimate biquadratic surface patch fits to range data. Their method assumes that additive range errors are of the form of a Student-t distribution with the degree of freedom between 1 and 2. Their efficient algorithm demonstrates being able to detect boundaries between different surfaces as well as ignore gross outlier data. Unfortunately, for our data a naïve implementation of their algorithm produces unreliable segmentations, as seen in Fig. 5.1. A more complete attempt based on later work by them [3] would likely yield better results.

Srikantiah [26] developed a pseudo-multi-scale method for segmenting meshes using surface curvature estimates found from local biquadratic surface patch fits. He first uses a region growing technique to group together surface points that all have the same mean and Gaussian curvatures, within a tolerance window. After finding all such large segments, he extracts segments that have only similar mean or Gaussian curvatures. This was in fact the first segmentation method we attempted to use on our data.

Unfortunately, we have a very frequent condition that occurs in our data that foils his approach: we have neighboring segments that have the same curvature but different orientations and often lack a well-sampled boundary. This results in points that are immediate neighbors having the same curvature estimates even though their normals point in different directions. Fig. 5.2 shows an example of this condition. For 2D curves, this condition may not happen often. For 3D surfaces, this condition need only happen with a single pair of points from the neighboring surfaces since a strict region growing method is used.

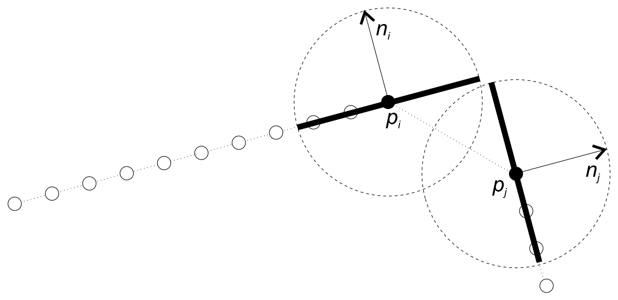


Figure 5.2: **Same curvature estimates, inconsistent orientations.** If p_i and p_j are neighboring points on a mesh, they may both have the same curvature estimates (*i.e.* zero here) even though their normal orientations clearly indicate they belong to different surfaces. The dashed circles represent the local neighborhood size used when estimating the curvature. The bold lines represent the local curve (surface) fit.

5.2.2 Spectral Clustering

A number of researchers have recently been examining spectral methods for segmentation, clustering, and related problems [17], [20], [21], [22], [25], [30], [32], [35]. In general, these methods require a symmetric affinity matrix $\mathbf{A}$ that defines a similarity measure between points p_i and p_j, often of the form

$$\mathbf{A}_{i,j} = \prod_{m=1}^{M} \left(\exp \left(- \frac{|| \, d_m \left(t_i, t_j \right) ||^2}{2{\sigma_m}^2} \right) \right) \tag{5.1}$$

where t_i is a set of attributes for point p_i, σ_m is a free parameter generally related to an estimate of the measurement noise, M is the number of distance measures used, and $d_m \left(\cdot, \cdot \right)$ is some distance measure.

Ideally, the affinity between points that belong in the same segment is one and the affinity between points not in the same segment is zero. Under these conditions, $\mathbf{A}$ is a block diagonal matrix. This means that, after possible row-column permutations, it has the following form (assuming there are two segments in this example)

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{Z} \\ \mathbf{Z}^T & \mathbf{C} \end{bmatrix} \tag{5.2}$$

where **B** and **C** are square matrices of ones and **Z** is a rectangular matrix of all zeros. In practice, **B** and **C** must contain a large number of large affinity values and **Z** must have only a few large values and/or consist of small values.

The eigenvalue corresponding to the largest eigenvector of **A** will contain ones for all points that belong to the largest cluster and zeros elsewhere, as explained by Weiss [32]. Sarkar and Boyer [22] and Perona and Freeman [21] use the first eigenvalues of **A** for change detection and a simplification of **A** for segmentation, respectively. Using this formulation directly results in what Shi and Malik call the *average association.* This method tends to be effective at selecting highly coherent clusters, but does not always yield optimal segmentations in practice.

Several researchers use a modified form of the affinity matrix to improve the segmentation quality. Shi and Malik [25] use the degree of **A**

$$\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{i,j} \tag{5.3}$$

to normalize **A** into the following generalized eigenvector problem:

$$(\mathbf{D} - \mathbf{A})y_i = \lambda_i \mathbf{D} y_i \tag{5.4}$$

By examining the second smallest eigenvector of $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, they are able to determine the first segment. They continue to recursively partition the graph until the normalized cut value rises to a threshold value. This value is defined as

$$Ncut\,(B, C) = \frac{cut\,(B, C)}{assoc\,(B, V)} + \frac{cut\,(B, C)}{assoc\,(C, V)} \tag{5.5}$$

where B and C form a partition of the full set of points V. The cut value is defined as

$$cut\left(B,C\right) = \sum_{b \in B, c \in C} \mathbf{A}_{b,c} \tag{5.6}$$

and the association as

$$assoc\left(B,V\right) = \sum_{b \in B, v \in V} \mathbf{A}_{b,v} \tag{5.7}$$

This formulation favors breaking the set of points into balanced regions. This method is called *normalized cuts*.

Weiss [32] recommends combining normalized cuts with a second method to ease the extraction of segments after k eigenvectors are found. Ng *et al.* [20] use a slightly different normalization scheme that they claim can produce more reliable segmentations when the degree $\sum \mathbf{D}_{i,i}$ of different segments varies greatly across segments. Kannan *et al.* [17] offer an in-depth mathematical analysis of spectral clustering methods, and Wang and Siskind [30] propose an alternative but similar method called *ratio cuts*.

Yu *et al.* [35] have recently adapted normalized cuts for use with unorganized range points before surface reconstruction. Their segmentation objective is to produce an over-segmentation of objects in a room. The segments are then grouped manually to form objects such as lamps, walls, and furniture. For their similarity measure, they define the following asymmetric anisotropic proximity distance

$$d_{yu\_dist\_assym}\left(p_i, p_j\right) = \sqrt{E\left(v_{i,j} \circ n_i\right)^2 + ||v_{i,j} - \left(v_{i,j} \circ n_i\right) n_i||^2 / E} \tag{5.8}$$

where $v_{i,j} = p_i - p_j$, "$\circ$" is the dot product operator, $||\cdot||$ is the standard 2-norm, n_i is the estimated surface normal of point p_i, and $E \geq 1$ is a weighting parameter. In the first pass of their multi-scale segmentation, they use their adapted normalized

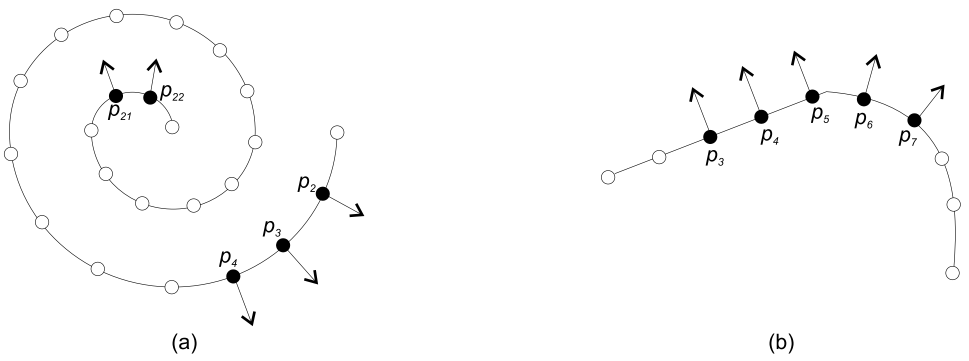


Figure 5.3: **Affinity transitivity.** (a) A spiral shape that would be grouped as one segment using normalized cuts. (b) A shape that would be split into two segments between points p_5 and p_6.

cuts algorithm with the following similarity measures:

$$d_{yu\_norm}\left(t_i, t_j\right) = \cos^{-1}\left(n_i \circ n_j\right) \tag{5.9}$$

$$d_{yu\_dist}\left(t_i, t_j\right) = max\left(d_{yu\_dist\_assym}\left(p_i, p_j\right), d_{yu\_dist\_assym}\left(p_j, p_i\right)\right) \tag{5.10}$$

Eqn. 5.9 indicates that the estimated normals of two points must be similar for those points to have a high affinity. Eqn. 5.10 compares the distance between the two points, but assigns a higher penalty to distance in the direction of the normal vectors, according to the choice of E in Eqn. 5.8. Eqns. 5.9 and 5.10 are used as the distance measures in Eqn. 5.1 to calculate the affinity value.

As was done by Shi and Malik, Yu *et al.* make their affinity matrix more sparse by only retaining a portion of the non-zero values for points not very close to each other. They have found that this makes the eigenvector calculations happen faster. After segmentation, Yu *et al.* reconstruct meshes using `crust`, a predecessor of `cocone` (see Chapter 4).

These spectral clustering methods assume that a property which we call transitivity is desired. This property states that if $\mathbf{A}_{i,j} \gg 0$ and $\mathbf{A}_{j,k} \gg 0$ then points p_i and p_k belong to the same segment, even if $\mathbf{A}_{i,k}$ is small. Fig. 5.3a shows an extreme example of this principle. Even though affinity $\mathbf{A}_{4,21}$ is low (assuming Eqns. 5.9 and 5.10 are used), points p_4 and p_{22} will be grouped into the same segment because affinities $\{..., \mathbf{A}_{2,3}, \mathbf{A}_{3,4}, ..., \mathbf{A}_{21,22}, ...\}$ are all high. On the other hand, the shape in 5.3b would be segmented between points p_5 and p_6 because there is a sudden jump in the normal direction between these points. One of the strengths of formulations similar to normalized cuts is that they are able to exploit transitive relationships such as these better than other formulations such as average association.

5.3 Our Segmentation Algorithm

We have chosen to use a modified version of normalized cuts for segmenting the meshes reconstructed via `cocone` (see Chapter 4). In doing so, we have learned a number of valuable lessons which we will discuss in this section.

The first lesson is that the choice of affinity function is very important in determining the quality and reliability of segmentations. After exploring a number of less heuristic affinity measures (Section 5.4), we found that only formulations similar to those of Yu *et al.* (Eqns. 5.9 and 5.10) produced reliable segmentations for our data. The final formulation that we have chosen is based on the same core ideas,

$$\mathbf{a}_{i,j} = exp\left(-\frac{\cos^{-1}\left(n_i \circ n_j\right)}{2\left(\frac{\pi}{180^o}\sigma_n\right)^2}\right) exp\left(-\frac{d_{yu\_dist\_assym}\left(p_i, p_j\right)}{2r^2}\right) \tag{5.11}$$

$$\mathbf{A}_{i,j} = min\left(\mathbf{a}_{i,j}, \mathbf{a}_{j,i}\right) \tag{5.12}$$

where $d_{yu\_dist\_assym}\left(\cdot,\cdot\right)$ is the same as was defined in Eqn. 5.8, E is set to 2, and we choose $r = 512mm$ and $\sigma_n = 10^o$. Any points that are more than r millimeters

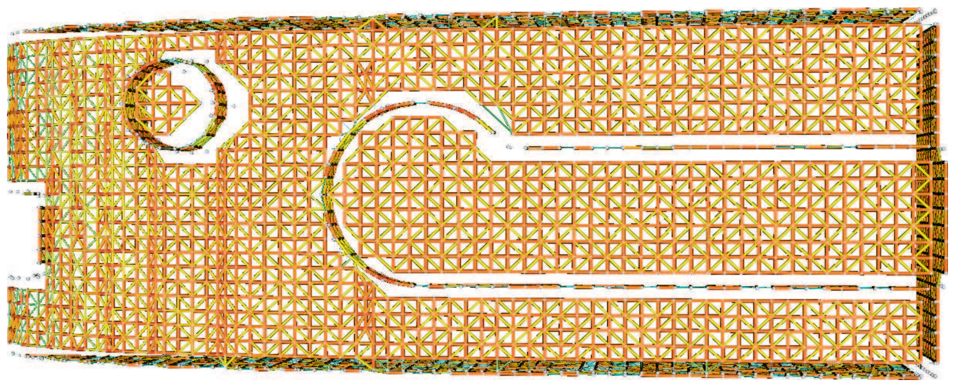


Figure 5.4: **Unsegmented disjoint regions using normalized cuts.** An object with disjoint segments that does not get properly segmented using normalized cuts due to a poor choice in affinity measure and too few Lanczos iterations allowed. Lines connecting points represent non-zero affinities between those points. The thickness of the line is proportional to the affinity value.

apart have their affinities set to zero. We also only retain 25% of the affinities between points separated by more than $175mm$ and we discard any affinities that are less than 0.1. By using min instead of max to compare $\mathbf{a}_{i,j}$ and $\mathbf{a}_{j,i}$, we were able to slightly improve the reliability of the segmentation for our data.

The second problem area for us relates to the number of iterations required for the Lanczos eigensolver [14, pp. 475–504], [38]. We based our implementation on one adapted from Shi and Malik by Doron Tal for image segmentation. In that implementation, only 25 iterations were allowed. Although so few iterations may be enough for image segmentation, we found that we needed ten times as many for our mesh segmentation. We saw cases where normalized cuts was unable to separate fully-disjoint segments (*i.e.* where $\mathbf{A}$ is truly block-diagonal), such as is shown in Fig. 5.4.

Additionally, for some affinity formulations we found that the `lapack` implementation of the eigensolver resulted in divisions by zero and other data corruption. In cases where the eigenvectors did not get corrupted, occasionally the eigensolver would be stuck in a loop oscillating between two eigenvector solutions without converging. We noticed these problems when we did not force the retention of the affinity values for immediate mesh neighbors.

5.4 Probabilistic Affinities

In Section 5.3, we described the final affinity functions that we employed in our recognition system. In this section, we will discuss two other similarity measures with which we experimented, and we will discuss the reasons we believe that they did not work as well as the more heuristic measure given by Eqns. 5.11 and 5.12.

In particular, we will examine $A_{pos}(\cdot,\cdot)$ and $A_{norm}(\cdot,\cdot)$ where

$$\mathbf{A}_{i,j} = min\left(A_{pos}(p_i,p_j)\,A_{norm}(p_i,p_j)\,,A_{pos}(p_j,p_i)\,A_{norm}(p_j,p_i)\right) \tag{5.13}$$

These distance measures consider the position of a point relative to the other's surface fit and the consistency of normal estimates, respectively. Each of these has its own σ parameter, σ_{pos}, σ_{norm}, σ_{dist}, and σ_{fit}, respectively. Fig. 5.5 gives an overview of what the position, normal, and distance affinities are measuring (simplified to 2D curves).

5.4.1 Position Affinity

Our position affinity, $A_{pos}(p_i,p_j)$, is proportional to the probability that point p_j was measured from the surface fit $S_i(u,v)$ of p_i, given that p_j was sampled from that surface. We make the simplifying assumption that in a local estimation neighborhood, the errors in the biquadratic fit are due to identical, independently distributed (IID)

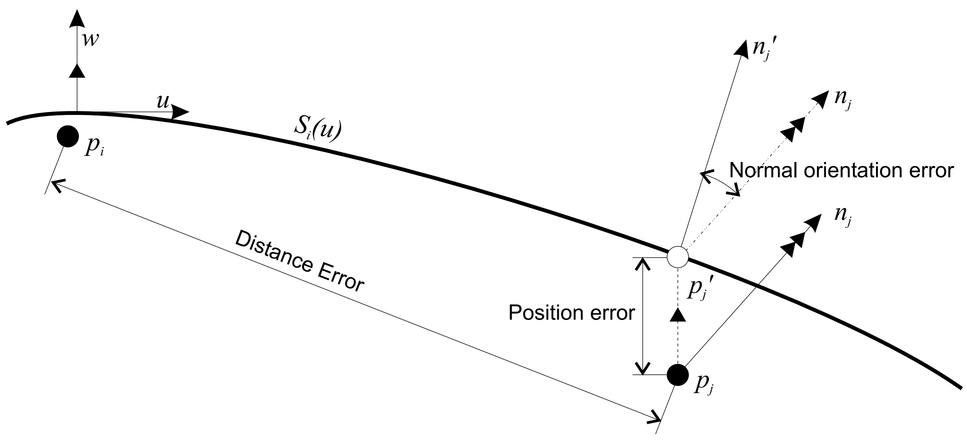
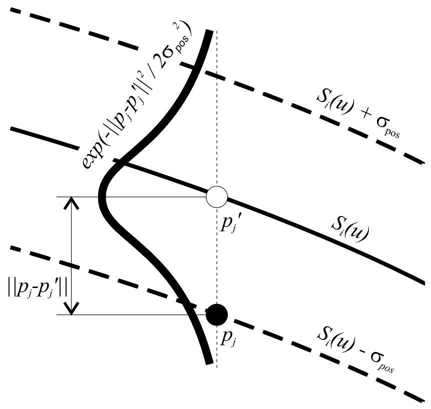


Figure 5.5: **General affinity framework.** The geometric framework (shown in 2D) from which the probabilistically-based $\mathbf{A}_{i,j}$ is derived given a surface estimate $S_i(u)$ from point p_i. p'_j is the projection of p_j onto curve $S_i(u)$ (or $S_i(u,v)$ in 3D). Solid equilateral triangles mark parallel lines. Note that the origin of the local (u,v,w) coordinate system for point p_i does *not* actually pass through point p_i in general (see Sec. 3.3).



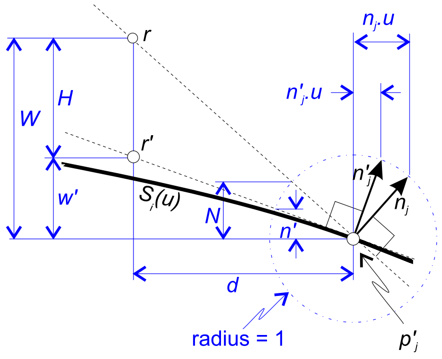


Figure 5.6: **Position affinity framework.** Position affinity framework. The dashed lines are the curve $S_i(u)$ displaced vertically by the distance between p_j and its projection, p'_j, onto $S_i(u)$. The Gaussian by which the position affinity is scaled is overlaid (thick line).

Figure 5.7: **Normal affinity framework.** Important objects in the derivation of the 2D normal orientation affinity. See section 5.4.2 for a more detailed explanation of the math in this figure.

Gaussian noise in the w-direction. It can be shown that the least squares fit we explained in Section 3.3 is an optimal estimate of the surface under those conditions. Furthermore, the mean squared error in the fit, ${\sigma_{pos}}^2$ serves as the optimal estimate of the variance of the Gaussian noise [27].

Based on that assumption, the measurement probability and the position affinity are given by:

$$P[p_j \text{ would be measured from } S_i(u,v)] \propto$$
$$A_{pos}\left(p_i, p_j\right) = \exp\left(-\frac{||p_j - {p'}_j||^2}{2{\sigma_{pos}}^2}\right) \quad (5.14)$$

where ${p'}_j = (p_j.u, p_j.v, S_i(p_j.u, p_j.v))$ is the projection of p_j onto surface S_i and $\triangle.u$ is the u component of point $\triangle$, *etc.* Fig. 5.6 is a pictorial description of the inputs to the exponential.

Although we might choose ${\sigma_{pos}}^2$ to be the surface fit error for S_i, as described above, this approach would yield poor results. Under our model, surface fit discontinuities are rare events and correspond to the edges of different underlying low order surfaces. At such discontinuities, the surface fit error is more likely to be driven by the edge effect rather than by sensor noise.

Under the assumption that the sensor noise characteristics do not change from one part of the surface to another, we instead calculated an average surface fit error across all points and used that global fit error. We found that this estimate was generally only slightly higher than the known variance of the Gaussian noise we added to our simulated data. Unfortunately, this approach yielded over-segmentation on low-noise scenes and under-segmentation on high-noise scenes. This is because the noise due to edge effects does not change with sensor noise. Instead we found that if we chose a single ${\sigma_{pos}}^2$ regardless of the actual sensor noise simulated, we had more consistent results.

When we were originally using random point selection to subsample the range points (Section 3.2.1), we only calculated affinities between points not much more distant than the furthest point that was used to estimate the biquadratic fit. As such, this positional affinity worked reasonably well at determining whether point p_j arose from the surface of p_i.

We knew from our experiments with Srikantiah's segmentation algorithm (Section 5.2.1) that orientation information is critical for segmenting our data. As discussed in Section 3.2.2, the random point selection results in overly-smooth biquadratic estimates. In particular, the normal estimates are smoothed to the point where too often there is no detectable discontinuity in orientation change. This drove us to make our biquadratic fit estimates much more locally. Upon doing so we began comparing points a long distance away from where the biquadratic fit was made. With even a little bit of noise, these fits tend to become highly unreliable outside the estimation radius, and our position error estimates thus also became unreliable.

These difficulties were finally overcome by abandoning the probabilistically-based position affinity and adopting the distance measure proposed by Yu *et al.* (Eqn. 5.8). It is our observation that this measure effectively makes a position error estimate based on a planar fit instead of a biquadratic one.

5.4.2 Normal Orientation Affinity

As was already mentioned, when we were developing our probabilistic position affinity function, we knew we would need to somehow compare the estimated surface orientations of points. Toward this end, we began to develop a probabilistic normal orientation affinity, $A_{norm}\left(p_i, p_j\right)$. It relates the probability that the normal found from the surface estimate of p_j would be measured from the surface of p_i at the projected point ${p'}_j$. We begin by first considering the 2D case.

As with $A_{pos}\left(p_i, p_j\right)$, we assume that errors in the biquadratic fit are due to IID Gaussian noise. Fig. 5.7 depicts the relationships between the vectors and measurements we will be analyzing and will be used without further referencing in this section. We also make the simplifying assumption that the curve normal, n_j for p_j was estimated by finding the vector perpendicular to the line segment connecting p_j and another measured point, r, which is d distance units away from p_j in the u direction. To simplify the drawing, p_j and ${p'}_j$ are shown as coincident. For non-coincident p_j and ${p'}_j$, r would move vertically with p_j.

From trigonometry, we can see:

$$n = n_j.u = g(w) = \frac{w}{\sqrt{d^2 + w^2}} \tag{5.15}$$

where $n_j.u$ is the u component of the normal n_j, $g(w)$ is a function of the deterministic variable w, and $g(W)$ is a function of the random variable W which we wish to invert:

$$g(W) = N = \frac{W}{\sqrt{d^2 + W^2}} \tag{5.16}$$

$$N^2(d^2 + W^2) = W^2 \tag{5.17}$$

$$N^2 d^2 = W^2(1 - N^2) \tag{5.18}$$

$$g^{-1}(N) = W = \frac{Nd}{\sqrt{1 - N^2}} \tag{5.19}$$

Likewise, for the normal calculated at the projected point ${p'}_j$ leads to the deterministic relation

$$g^{-1}(n') = \frac{n'd}{\sqrt{1 - n'^2}} \tag{5.20}$$

The derivative of g is given by

$$\frac{\partial g(w)}{\partial w} = \frac{1}{\sqrt{d^2+w^2}} - w^2(d^2+w^2)^{-3/2} \tag{5.21}$$

Under our current model W is a random variable defined as:

$$W = r.w - p_j.w \tag{5.22}$$

where $r.w$ and $p_j.w$ are random variables distributed as Gaussians with variance ${\sigma_{pos}}^2$ and means $r'.w$ and ${p'}_j.w$, respectively. W thus is distributed as a Gaussian with variance $2{\sigma_{pos}}^2$ and mean $w' = r'.w - {p'}_j.w$. $H = W - w'$ has the probability density function (pdf)

$$f_H(h) \propto exp\left(-\frac{h^2}{4{\sigma_{pos}}^2}\right) \tag{5.23}$$

giving us the pdf of W:

$$f_W(w) \propto exp\left(-\frac{(w-w')^2}{4{\sigma_{pos}}^2}\right) \tag{5.24}$$

In general, given a random variable, X and an invertible, monotonically non-decreasing transformation $Y = g(X)$, the pdf of Y is given by the following formula

$$f_Y(y) = f_X(g^{-1}(y))\left|\frac{\partial g(x)}{\partial x}\right|^{-1} \tag{5.25}$$

where $x = g^{-1}(y)$ [27, pp. 119–120]. Using this formula, we can find the pdf of N:

$$f_N(n) = f_W(g^{-1}(n))\left|\frac{\partial g(w)}{\partial w}\right|^{-1} \tag{5.26}$$

$$\propto \frac{exp\left(-\left(\frac{nd}{\sqrt{1-n^2}} - \frac{n'd}{\sqrt{1-n'^2}}\right)^2 \Big/ 4{\sigma_{pos}}^2\right)}{\left|\left(1/\sqrt{d^2+w^2}\right) - w^2(d^2+w^2)^{-3/2}\right|} = A_{norm,2D}\left(p_i, p_j\right) \tag{5.27}$$

Substituting $w = g^{-1}(n)$ back in, we define the 2D normal orientation affinity to be:

$$A_{norm,2D}\left(p_i, p_j\right) = \frac{exp\left(-\left(\frac{nd}{\sqrt{1-n^2}} - \frac{n'd}{\sqrt{1-n'^2}}\right)^2 \Big/ 4{\sigma_{pos}}^2\right)}{\left|\left(1/\sqrt{d^2+\frac{(nd)^2}{1-n^2}}\right) - \frac{(nd)^2}{1-n^2}\left(d^2+\frac{(nd)^2}{1-n^2}\right)^{-3/2}\right|} \tag{5.28}$$

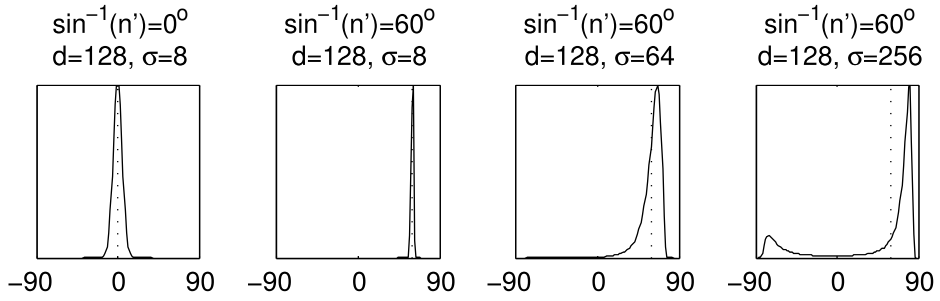


Figure 5.8: *angle* (n_j) **vs.** $A_{norm,2D}(p_i, p_j)$**.** 2D probabilistic normal orientation affinity distribution examples for several parameter values. The x-axis represents the angle from vertical of the estimated surface normal of point p_j. For reference, a vertical dotted line indicating the normal of the projected point is shown.

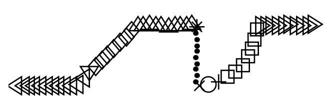


Figure 5.9: **Curve Segmentation Example Using Probabilistic Affinities.** Example segmentation of a curve using the probabilistic affinity measure defined in this section.

Several examples relating the angle n_j makes to the u-axis to the affinity value are shown in Fig. 5.8.

Fig. 5.9 shows an example segmentation of a curve using the probabilistic position and normal affinities described in this and the previous section. The example curve is $7,680mm$ long in the horizontal direction (a typical vehicle length) and contains 600 original points with $128mm$ 2D voxels. This yields approximately 10 points per $128\times128mm$ voxel, which is analogous to the 3D case where our vehicles typically have 100 points in a $128 \times 128 \times 128mm$ voxel. Independent isotropic Gaussian noise with $\sigma = 8mm$ is added to each point before quadratic curve estimation, corresponding to a medium noise level in our 3D data. Except for the quarter-circle region, the results shown in Fig. 5.9 seem to be promising.

Unfortunately, extending this 2D affinity to 3D is non-trivial and results in multiplying and adding several random variables whose distributions are of the form of Eqn. 5.27. We experimented with making a simplifying assumption that the components of the normal are separable in the u and v directions, but unsatisfactory segmentations resulted. In contrast, we found that Eqn. 5.9 provides an intuitive and effective affinity measure.

5.5 Segment Attribute Assignment

Once we have segmented the mesh using normalized cuts, we assign a number of attributes to each segment that are useful in the recognition process. In this section we will explain the attributes which we use as well as a few that we considered.

- **Area:** Among the most important segment attributes for our modelbase is surface area. To find the area contribution of an individual point, we first find the area of each mesh triangle using the well-known formula [13, pg. 1112]:

$$A_{i,j,k} = \frac{1}{2}||(p_i - p_j) \times (p_k - p_j)|| \tag{5.29}$$

  where p_i, p_j, and p_k are the vertices of the triangle. One third of each triangle's area is then assigned to each of its vertices.

- **PCA Attributes:** Using all of the points in the segment, we perform PCA again (see Section 3.3) to determine the principal axes, most notably the normal direction. If we think of PCA as fitting an ellipsoid to our data, then the square roots of the eigenvalues indicate the radii of the ellipsoid. We retain these values.

- **Thickness:** We define the thickness of the segment to be $\sqrt{\lambda_3/\lambda_1}$ where $\lambda_1 \geq \lambda_2 \geq \lambda_3$ are the PCA eigenvalues.

- **Elongation:** Similarly, we define the elongation of the segment to be $\sqrt{\lambda_2/\lambda_1}$. The elongation and planarity together give a gross estimate of the shape of the segment.

- **Centroid:** Finally, we calculate the centroid of the segment. Together with the normal direction, they provide attributes for making comparisons between pairs of object and model segments.

- **Other Attributes Considered:** We also considered using the mean and Gaussian curvature at the projection of the centroid onto a biquadratic fit of the surface. Since we have selected only relatively flat surfaces due to our range point subsampling process, these two features do not tend to be very descriptive.

  We considered using a distinctiveness measure as well that indicates how dissimilar a segment is to all segments from all other models:

$$\mathit{distinctiveness}\left(s, \{m_{i,j}\}\right) = 1 - \frac{1}{1 + \frac{1}{N} \sum_{i=1}^{N} \min_{j} d_u\left(s, m_{i,j}\right)} \tag{5.30}$$

  where s is the segment for which we wish to determine the distinctiveness, $\{m_{i,j}\}$ is the set of segments in the modelbase (not including any segments from the model from which s was taken), i is the model number, j is the segment number, $N+1$ is the number of models in the modelbase, and $d_u\left(\cdot,\cdot\right)$ is the

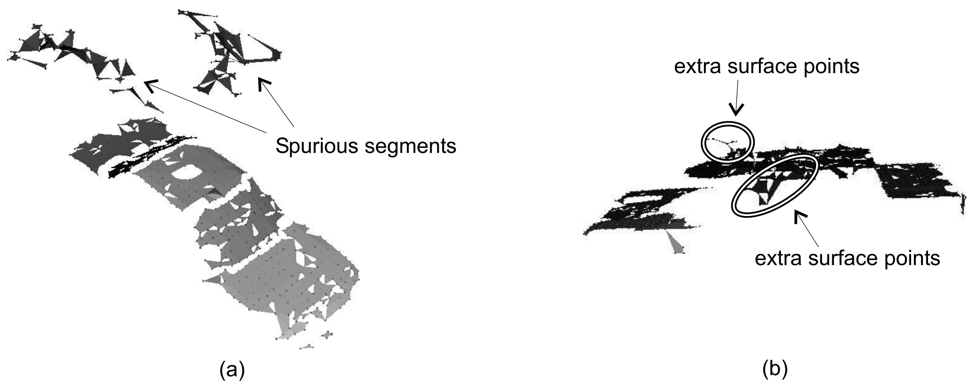


Figure 5.10: **Spurious segments and points.** (s) spurious segments for *sedan* in the *flyby* scene with no noise added. (b) extra surface points that have been grouped with the roof of the *tank* in the *flyby* where $\sigma = 4mm$.

unary feature distance to be described in Section 6.3 (Eqn. 6.3). Since our modelbase is currently so small, we did not find this measure to be very helpful.

5.6 Results

Using the affinity matrix defined in Eqn. 5.12 and rejecting any segment splits when $Ncut\,(B, C) \geq 0.1$ (Eqn. 5.5), we have segmented our test data sets. The results are shown in Figs. 5.11, 5.12, and 5.13 for the *circle*, *flyby*, and *unoccluded* scenes, respectively. In comparing the results, we will use as a baseline the first column in Fig. 5.13, corresponding to noiseless data taken from the *unoccluded* scene.

The most important differences in segmentations that we observe are the following:

1. **Area:** As a result of noise and/or occlusion, a very common difference in segments is that they have a smaller area than those from the baseline segmentation. For example, the roof of the sedan in the third row and last column of Fig. 5.11 has chunks missing due to occlusion and noise in the data. A more

severe version of differences in area are when too little of a surface is seen for a segment to even be created at all. For example, the two sides of the roof are not present for *obj1* in row 2, column 5 of Fig. 5.12.

2. **Spurious Segments:** Although the combination of pruning high aspect ratio triangles and using normalized cuts generally removes the clutter from the final results, sometimes spurious segments get formed as shown in Fig. 5.10a. Fortunately, those spurious segments that do get created rarely confuse our recognition engine.

3. **Over-segmentation:** One of the most common problems is with over-segmentation. Compare for example *obj1* (2^{nd} row) with $\sigma = 16$ (column 5) in the *circle* scene (Fig. 5.11) to the three roof segments in the baseline segmentation (row 2, column 1, Fig. 5.13).

4. **Under-segmentation:** Occasionally, surface regions that are split in the baseline object remain as one segment in another data set. This event is actually quite rare with our data.

5. **Misaligned segmentation:** We noticed that especially with the *tank* that a few extra points that really belong to one segment get grouped with another segment (see Fig. 5.10b).

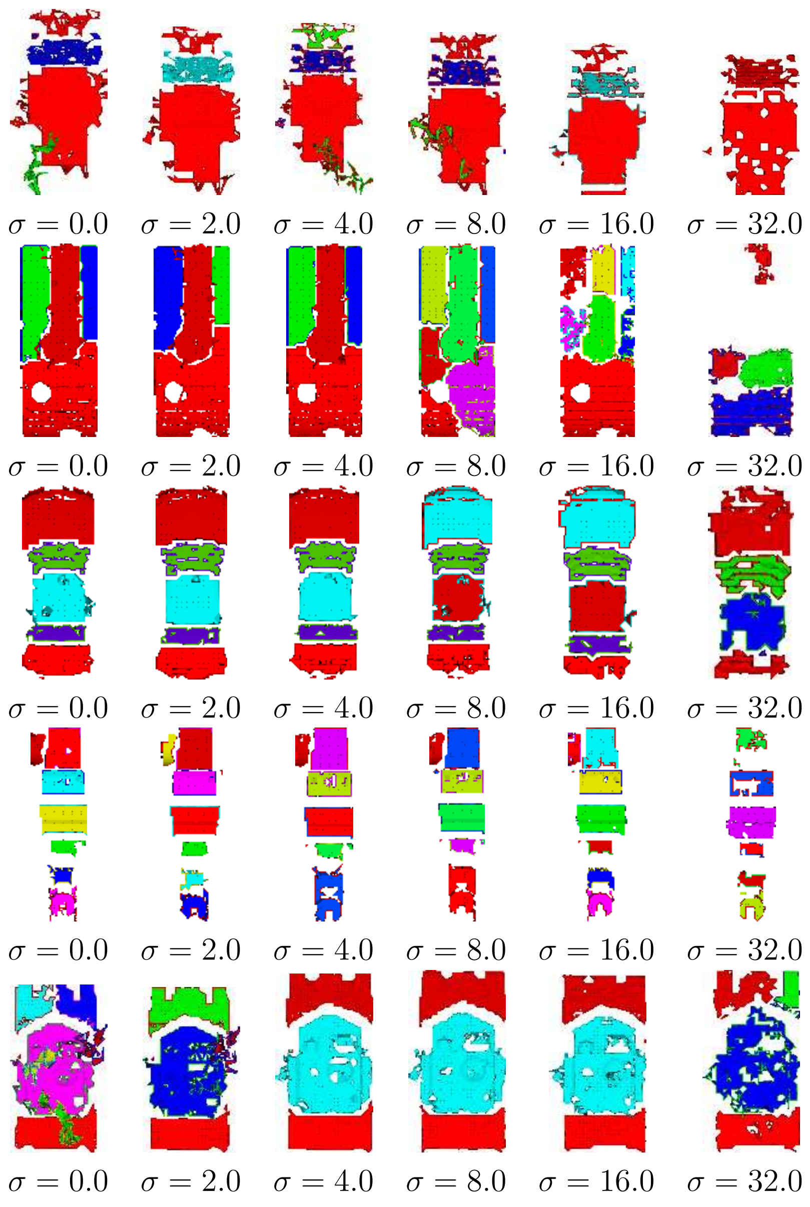

$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$

Figure 5.11: **Segmentations of the *circle* scene.** The rows contain the segmentation results of *earthmover*, *obj1*, *sedan*, *semi*, and *tank*, respectively for each of the synthetic noise levels.

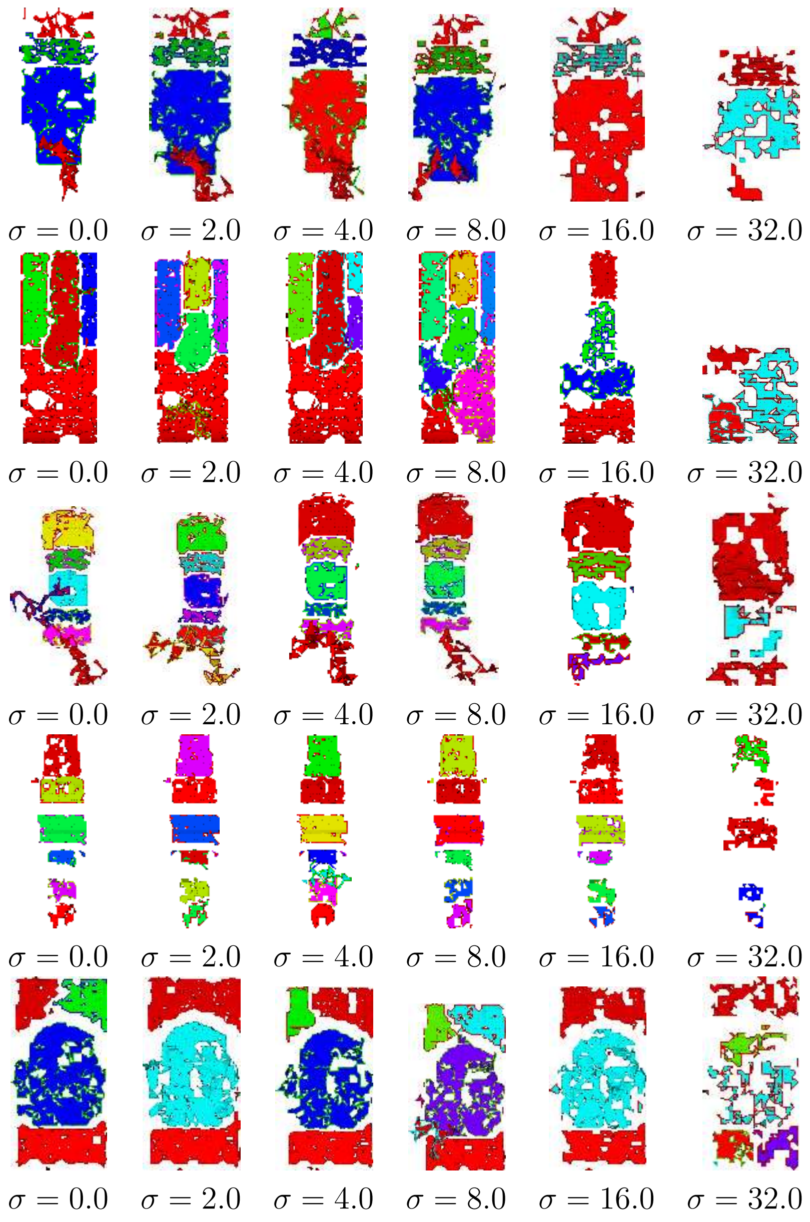

$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$
$\sigma = 0.0$ $\sigma = 2.0$ $\sigma = 4.0$ $\sigma = 8.0$ $\sigma = 16.0$ $\sigma = 32.0$

Figure 5.12: **Segmentations of the *flyby* scene.** The rows contain the segmentation results of *earthmover*, *obj1*, *sedan*, *semi*, and *tank*, respectively for each of the synthetic noise levels.

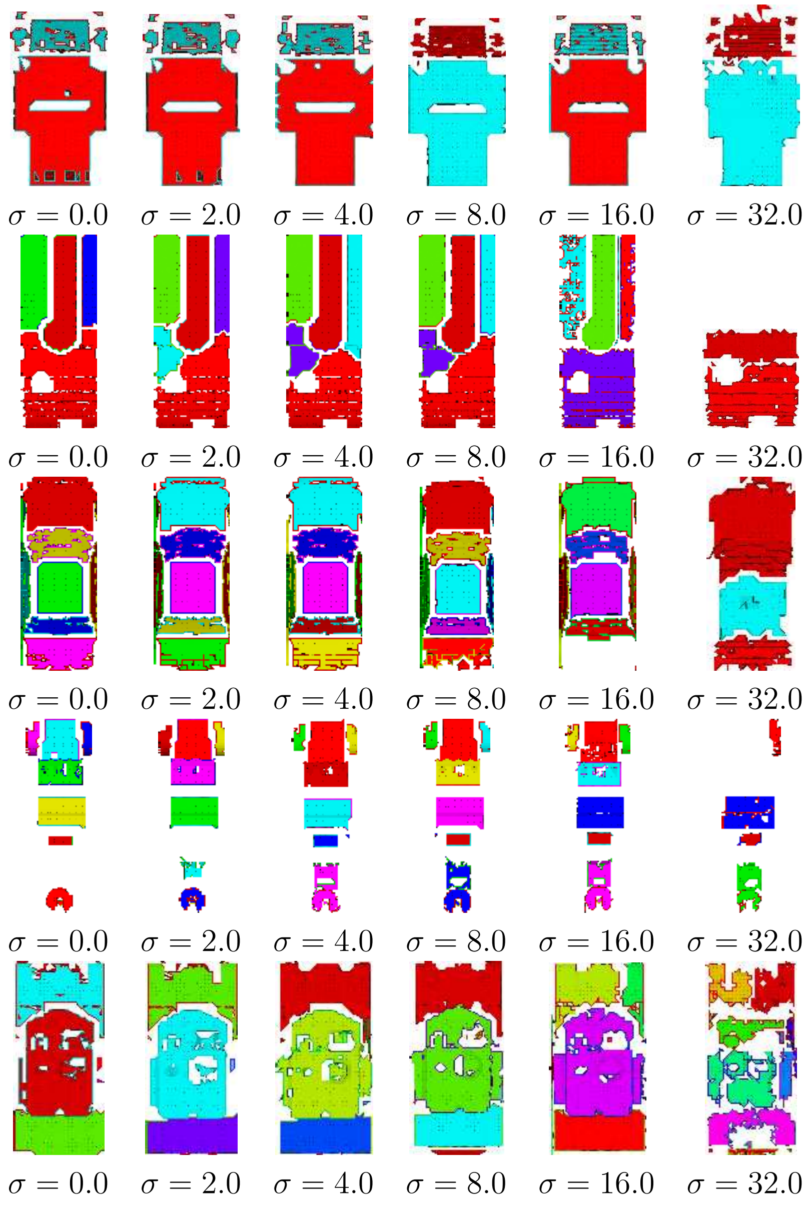


Figure 5.13: **Segmentations of the *unoccluded* scene.** The rows contain the segmentation results of *earthmover*, *obj1*, *sedan*, *semi*, and *tank*, respectively for each of the synthetic noise levels.

CHAPTER 6

GRAPH MATCHING AND OBJECT RECOGNITION

As has been mentioned and implied in previous chapters, we do our final object recognition by attempting to match up scene object segments produced by the algorithm described in Chapter 5 with modelbase segments. In Section 6.1 we give an overview of recognition systems used by other researchers. We then describe how graph matching is done in Section 6.2. In Section 6.3, we describe our matching error measure. We discuss our recognition results in Section 6.4.

6.1 Prior Work

Object recognition techniques are commonly divided into two classes: those that use global features and those that use local features to do matching. Global methods use statistics and features based on the entire shape. As an example, one might say an object is a car because it has the right silhouette when seen from a side view. Local feature methods use attributes from regions of the shape and generally encode spatial relationships between these regions. In this case, one might recognize a car by seeing a hood connected to sides which are connected to the windows which are connected to a roof.

Campbell and Flynn provide an excellent overview of recent recognition methods in [4]. In this section, we will summarize some of the methods described there as well some additional techniques we have considered.

One popular category of global feature methods has been termed “appearance-based recognition.” Typically only a single view has been taken of an object. Feature information (*e.g.* hue, range, curvature, *etc.*) are sampled on the surface in a topologically uniform manner. The feature values become coordinates in some high dimensional space. Principal Component Analysis (Section 3.3) on a representative set of views using a model can then be employed to create a basis. A given view is then recognized by projecting its feature vector onto the basis and determining which model yields the lowest projection error [4]. We have chosen not to use an appearance-based technique for our system for two reasons. First, we expect to use many object views, not just one since any single view will have significant clutter. Second, appearance-based techniques often perform poorly when occlusions and highly-variable sampling conditions are present.

Another related set of global feature techniques are the Extended Gaussian Image and Spherical Attribute Image (SAI) [15]. In these techniques, a geometrically and topologically semi-uniform mesh is fitted to a surface representation of an object and a curvature-based measurement is made at each mesh vertex. Because a vector is a one dimensional list and we have a 2D manifold, multiple feature vectors must be created from these measurements. A particular feature vector encodes the shape given a particular orientation. A modelbase is created offline with a sampling of the possible orientations of each model. An exhaustive recognition can then be performed efficiently by taking the dot product of the scene object’s feature vector with each vector in the database. Herbert *et al.* use a feature vector of approximately 1000 elements and have 3000 feature vectors per model [15]. They have shown that this technique works for certain types of occlusions. We considered using this approach for our recognition engine; however, the mesh regularization step is both non-trivial and critical to the success of the method. Additionally, our scenes tend to have a combination of many small occluded patches with several large-area occlusions. Additional work would be required to fully evaluate the SAI approach in our environment.

One of the reasons for using local features in a recognition system has already been mentioned: occlusion. A missing part in an appearance-based scheme may sufficiently perturb the feature vector so that the entire shape seen does not look much like the entire model. Additionally, ambiguities can arise that complicate the usage of the SAI when many occluded regions exist. A common way of using local features is to identify a (small) number of salient regions of the object. For example, one could identify crease edges. We identify surface segments, as has been described in Section 5. These segments are then matched up with ones in the modelbase, checking the similarity of object-to-model segment pairings and the spatial relationships between the pairings. In the next section, we describe one method for doing this matching. Some other methods include those describe by Wong *et al.* [33], Srikantiah [26], and Chen and Kak [5].

6.2 Graph Matching

When using graph matching techniques for object recognition, we represent objects as attributed graphs where each object segment corresponds to a graph node and the links between nodes signify explicit relationships amongst the segments. As a

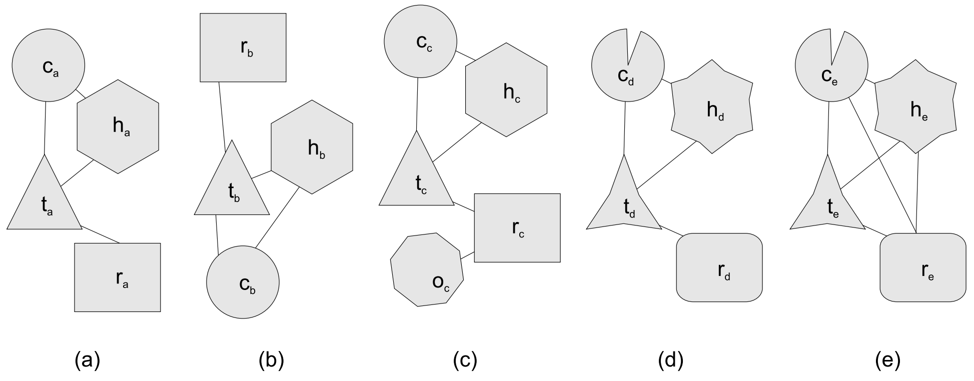


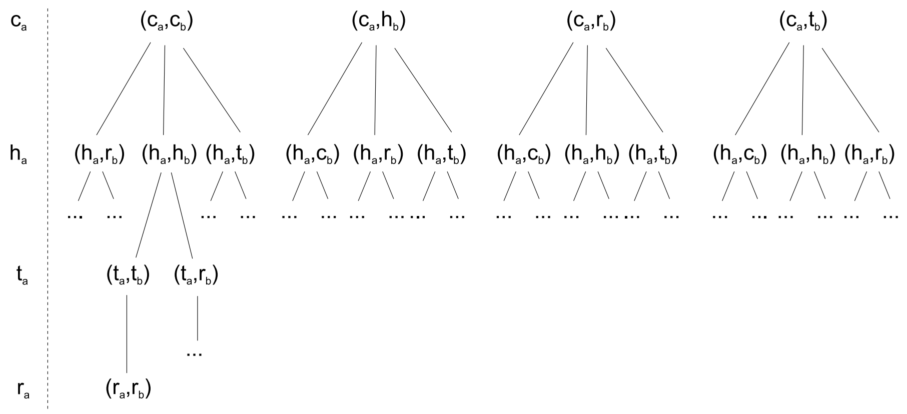
Figure 6.1: **Five attributed graphs.**

running example in this section, we will use the graphs in Fig. 6.1. The shaded shapes are the nodes where "triangle", "circle", *etc.* are their attributes. The lines connecting them represent the links in the graph. For the sake of simplicity, we consider the only attribute of a link to be whether it is present or not (in this section). Our graph matching algorithm is based on the work of Flynn and Jain [12] and Srikantiah [26].

A *complete graph match* is a set of ordered pairs of segments such that the pairs contain one segment from each graph, the two segments in each pair are the same, and all of the links between the pairs are the same. For example, $\{(c_a, c_b)$, (h_a, h_b), (r_a, r_b), $(t_a, t_b)\}$ is a complete match between the graphs shown in Fig. 6.1a and 6.1b.

An *incomplete graph match* is one where only some of the nodes are matched up. For example, $\{(c_a, c_c)$, (h_a, h_c), $(t_a, t_c)\}$ is an incomplete match because segments o_c and r_c from graph c and segment r_a from graph a are not matched up with any other segments.

An *inexact graph match* is a graph match in which the nodes and links may not exactly match up. For example, $\{(c_a, c_d)$, (h_a, h_d), (r_a, r_d), $(t_a, t_d)\}$ is an inexact match



c_a
h_a
t_a
r_a
(c_a,c_b)
(c_a,h_b)
(c_a,r_b)
(c_a,t_b)
(h_a,r_b) (h_a,h_b) (h_a,t_b)
(h_a,c_b) (h_a,r_b) (h_a,t_b)
(h_a,c_b) (h_a,h_b) (h_a,t_b)
(h_a,c_b) (h_a,h_b) (h_a,r_b)
(t_a,t_b) (t_a,r_b)
(r_a,r_b)

Figure 6.2: **Tree-based graph matching example.** Consider the segments in Fig. 6.1a and Fig. 6.1b. This is a partial match tree. The full match tree would contain a path for every possible matching of segments from graph a to those in graph b. Along the left side of the figure is the object segment being matched for the given tree level.

because the nodes in graphs b are slightly different than those in a and there are two extra links between h_d and r_d and between c_d and r_d.

A *complete graph* is one in which all nodes are connected to all other nodes, such as is shown in Fig. 6.1e.

A *tree-based* matching algorithm is one that recursively attempts to assign nodes from an object to nodes from a given model. Fig. 6.2 shows a pictorial representation of part of a match tree for the segments in Fig. 6.1a and Fig. 6.1b. These algorithms are exponential in complexity.

For the sake of implementation simplicity, we have created a depth-first, tree-based, incomplete, inexact graph match algorithm given a set of object segments $S = \{s_0, s_1, ...\}$ and a modelbase. Since our modelbase is small, we attempt to match the object to each model and we rank the results by match score. Before beginning the matching process, we sort the object segments by area, with the largest segment being matched first.

At any particular level in the match tree, we evaluate the match error for each of the branches going to the next level. To maintain runtime performance, we allow for only recursing on the B best branches. We also will consider only the X best object segments, limiting the depth of the tree. For our experiments, we have chosen $B = 5$ and $X = 15$.

6.3 Match Error Measure

We will now explain the match error calculations. We begin these calculations by determining the *unary match error* between a set of x pairs of segments $\{(s_1, m_1), ..., (s_x, m_x)\}$ that have been tentatively matched up:

$$E_U(s_i, m_i) = 1 - exp\left(-\, {d_u}^2(s_i, m_i)\right) \tag{6.1}$$

$$C_U(\{(s_1, m_1), ..., (s_x, m_x)\}) = \sum_{i=1}^{x} E_U(s_i, m_i) \tag{6.2}$$

where $E_U(\cdot, \cdot)$ is the pairwise unary match error, $C_U(\cdot)$ is the cumulative unary match error, and $d_u(\cdot, \cdot)$ is the *unary feature distance* and is given by

$$d_u(s_i, m_i) = \sqrt{{d_a}^2(s_i, m_i) + {d_t}^2(s_i, m_i) + {d_e}^2(s_i, m_i)} \tag{6.3}$$

where $d_a(\cdot, \cdot)$ is the *unary area match error*, $d_t(\cdot, \cdot)$ is the *unary thickness match error*, and $d_e(\cdot, \cdot)$ is the *unary elongation match error*. To limit the amount of time spent exploring portions of the match tree that are unlikely to result in good matches, we halt our recursive search any time the pairwise unary match error $E_U(\cdot, \cdot)$ is greater than some threshold (0.6 for our experiments).

When comparing object and model segment areas, we wish to penalize for multiplicative differences, thus we have chosen

$$d_a\left(s_i, m_i\right) = -\,log_{(1.0+f_a)}\left(\frac{min\left(a_{s_i}, a_{m_i}\right)}{max\left(a_{s_i}, a_{m_i}\right)}\right) \tag{6.4}$$

where a_{s_i} is the area of segment s_i and $100f_a$ is the percentage change in area required to produce one feature distance unit. We are currently using $f_a = 1$.

In comparing thickness and elongation, we wish to penalize for additive instead of multiplicative differences. Thus

$$d_t\left(s_i, m_i\right) = w_t|t_{s_i} - t_{m_i}| \tag{6.5}$$

$$d_e\left(s_i, m_i\right) = -w_e\, ln\left(1 - |e_{s_i} - e_{m_i}|\right) \tag{6.6}$$

where w_t and w_e are weighting factors (chosen to be 10 and 1, respectively) and t_{s_i} and e_{s_i} are the thickness and elongation of segment s_i, as defined in Section 5.5.

After determining how well the individual pairs of segments match up, we wish to evaluate their spatial relationships using the *orientation match error* and *centroid distance match error*. The orientation match error is given by

$$E_O\left(s_i, s_j, m_i, m_j\right) = \frac{2}{\pi}\left|\left(cos^{-1}\left(n_{s_i} \circ n_{m_i}\right)\right) - \left(cos^{-1}\left(n_{s_j} \circ n_{m_j}\right)\right)\right| \tag{6.7}$$

$$C_O\left(\{(s_1, m_1), ..., (s_x, m_x)\}\right) = \sum_{i=1}^{x}\sum_{j=1}^{x} E_O\left(s_i, s_j, m_i, m_j\right) \tag{6.8}$$

and it ensure that the difference in surface normals at the segment centroids match up. Similar to unary match error, we halt the recursion any time the value for $E_O\left(s_i, s_j, m_i, m_j\right)$ rises above some threshold (0.3 for our experiments).

Similarly, the centroid distance match error E_D compares the separation of segment centroids, as follows

$$d_R\left(a,b\right) = \begin{cases} 0 & \text{if } a=0 \text{ and } b=0 \\ \left|a-b\right| / \, max\left(a,b\right) & \text{otherwise} \end{cases} \tag{6.9}$$

$$C_D\left(\{(s_1,m_1),...,(s_x,m_x)\}\right) = \sum_{i=1}^{x}\sum_{j=1}^{x} d_R\left(||c_{s_i}-c_{s_j}||, ||c_{m_i}-c_{m_j}||\right) \tag{6.10}$$

where c_{s_i} is the centroid of segment s_i and x is the number of segments currently matched up. Any time $d_R\left(||c_{s_i}-c_{s_j}||, ||c_{m_i}-c_{m_j}||\right)$ is greater than some threshold (0.3 for our experiments), we halt the match tree recursion.

Finally, we wish to include a penalty for only matching small parts of the model. The *cumulative area error* is given by

$$C_A\left(\{(s_1,m_1),...,(s_x,m_x)\}\right) = min\left(1, \left|1-\sum_{i=1}^{x} a_{s_i} \Big/ \sum_{i=1}^{X} a_{m_i}\right|\right) \tag{6.11}$$

where X is the total number of segments in the model.

We now combine the four top-level match error scores into a single cumulative error measure,

$$E\left(M\right) = \frac{1}{x} C_U\left(M\right)^{e_u} C_O\left(M\right)^{e_o} C_D\left(M\right)^{e_d} C_A\left(M\right)^{e_a} \tag{6.12}$$

where $M = \{(s_1,m_1),...,(s_x,m_x)\}$ is the match being evaluated, and where we have chosen the weighting terms $e_u = e_o = e_d = e_a = 1$. This error value ranges from 0 to 1.

We sometimes have spurious segments in the object being recognized relative to a given model. These segments can exist for the following reasons:

1. The object is not an example of the model (*i.e.* we are trying to match a car with a tank).

2. The object was over-, under-, or incorrectly segmented.

3. Pieces of foliage looked enough like a vehicle to confuse the local surface fitting, surface reconstruction, and surface segmentation.

Because of these conditions, we allow for *NULL segment matches*, denoted $\{s_i, NULL\}$. Since a non-existent segment has no attributes, we set the values of Eqn. 6.1, 6.7, and 6.9 to zero any time a NULL segment is involved. We also set a_{m_i} to zero in Eqn. 6.11 to penalize for not matching segment s_i up with anything.

6.4 Results

To test our recognition engine, we took each of our objects and placed them in the *circle*, *flyby*, and *unoccluded* scenes with additive Gaussian noise standard deviations of 0, 2, 4, 8, 16, and 32mm. Renderings of the segmentation results that feed into the recognition process are shown in Figs. 5.11, 5.12, and 5.13.

We then generated the range data for these scenes (see Chapter 2), estimated local surface properties at a subset of the range points (see Chapter 3), reconstructed the surface meshes (see Chapter 4), and segmented the meshes (see Chapter 5). For each of the five objects, we selected the *unoccluded* scene where no noise was added and created a modelbase out of their segments. We will hereafter refer to these entries in the modelbase as “models”. We then computed the match score for each of the data sets generated to each of the models in the modelbase,

$$S(M) = 1 - E(M) \tag{6.13}$$

where $S(\cdot)$ is the match score and M is a given object-to-model match.

Tables 6.1, 6.2, 6.3, 6.4, and 6.5 give the recognition results for *earthmover*, *obj1*, *sedan*, *semi*, and *tank*, respectively. The entropy given for each set of matches is an indication of the uncertainty in the recognition results for a given data set with each of the five models. It is defined as follows:

$$s\left(\{M_1, ..., M_N\}\right) = \sum_{k=1}^{N} S\left(M_k\right) \tag{6.14}$$

$$entropy\left(\{M_1, ..., M_N\}\right) = -\sum_{k=1}^{N} \begin{cases} \frac{S(M_k)}{s(M_k)}\, log_2\left(\frac{S(M_k)}{s(M_k)}\right) & \text{if } s\left(M_k\right) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.15}$$

where $\{M_1, ..., M_N\}$ is the set of matches to each model given an object assuming there are N models. The entropy varies from zero to $log_2\left(N\right)$. A value of zero indicates complete certainty in the recognition result, and the maximal value indicates that the object looks like every model as much the others. Given five models, the entropy may vary from zero to approximately 2.3219.

We saw the best results with *sedan* and *earthmover*. Both were recognized correctly 100% of the time, but *sedan* had the lowest entropies. *Semi* also produced very promising results, only failing the recognition for the highest noise level. When the engine failed to place it first, twice the correct match was its second guess and once its third. In all three of these cases, the entropy was quite high. In general, the entropies were high for all but *sedan*.

Across all of the objects, we found that errors in area (including non-existent segments) were a significant factor in 10 of the 19 incorrect recognitions. Neither under-segmentation nor spurious segments contributed significantly to any of the errors, but over-segmentation (of the scene relative to the model) was a major factor in 12 of the erroneous recognitions. Misaligned segments contributed only to problems with the *tank*, where they participated in four of the errors, mostly in the data sets from the *flyby* scene.

We found that if we modified the affinity calculations to reduce the number misaligned segmentations that we greatly exacerbated the over-segmentation problem for all of the rest of the data.

Scene	**Noise Level**	**Score for Correct Match**	**Match Rank** (lower is better)	**Entropy** (uncertainty)
circle	0	0.824	1	2.16
circle	2	0.764	1	2.16
circle	4	0.819	1	2.19
circle	8	0.865	1	2.16
circle	16	0.797	1	2.15
circle	32	0.777	1	2.12
flyby	0	0.775	1	2.17
flyby	2	0.757	1	2.17
flyby	4	0.771	1	2.17
flyby	8	0.747	1	2.17
flyby	16	0.707	1	2.16
flyby	32	0.482	1	1.94
unoccluded	0	1.000	1	2.03
unoccluded	2	0.921	1	2.07
unoccluded	4	0.905	1	2.08
unoccluded	8	0.827	1	2.11
unoccluded	16	0.825	1	2.11
unoccluded	32	0.824	1	2.15
Recognition rate: 100%				

Table 6.1: **Recognition results for *earthmover*.** The object from the given scenes and noise levels was compared with the noiseless *unoccluded* scene for all five objects. The noise level is given in millimeters and the match score varies from 0 to 1 (with 1 being a perfect match). A match rank of 1 means the correct match was the first one while a rank of 5 means the recognizer thought that all other models looked more like object than the correct model. The entropy was calculated over the attempts to match against the five models.

Scene	**Noise Level**	**Score for Correct Match**	**Match Rank** (lower is better)	**Entropy** (uncertainty)
circle	0	0.756	2	1.86
circle	2	0.811	1	1.95
circle	4	0.816	1	1.96
circle	8	0.662	1	2.27
circle	16	0.468	4	2.25
circle	32	0.241	5	2.26
flyby	0	0.796	1	1.94
flyby	2	0.561	1	2.28
flyby	4	0.730	1	2.23
flyby	8	0.272	5	2.26
flyby	16	0.281	5	2.29
flyby	32	0.000	5	1.96
unoccluded	0	1.000	1	1.89
unoccluded	2	0.810	1	2.20
unoccluded	4	0.726	1	2.23
unoccluded	8	0.794	1	2.21
unoccluded	16	0.839	1	2.07
unoccluded	32	0.179	4	1.93
Recognition rate: 64%				

Table 6.2: **Recognition results for *obj1*.** The object from the given scenes and noise levels was compared with the noiseless *unoccluded* scene for all five objects. The noise level is given in millimeters and the match score varies from 0 to 1 (with 1 being a perfect match). A match rank of 1 means the correct match was the first one while a rank of 5 means the recognizer thought that all other models looked more like object than the correct model. The entropy was calculated over the attempts to match against the five models.

Scene	**Noise Level**	**Score for Correct Match**	**Match Rank** (lower is better)	**Entropy** (uncertainty)
circle	0	0.641	1	1.64
circle	2	0.723	1	1.65
circle	4	0.767	1	1.63
circle	8	0.589	1	1.69
circle	16	0.507	1	1.72
circle	32	0.486	1	1.71
flyby	0	0.683	1	1.61
flyby	2	0.576	1	1.68
flyby	4	0.662	1	1.62
flyby	8	0.602	1	1.68
flyby	16	0.447	1	1.72
flyby	32	0.291	1	1.71
unoccluded	0	1.000	1	1.46
unoccluded	2	0.937	1	1.50
unoccluded	4	0.894	1	1.52
unoccluded	8	0.918	1	1.51
unoccluded	16	0.666	1	1.67
unoccluded	32	0.451	1	1.70
Recognition rate: 100%				

Table 6.3: **Recognition results for *sedan*.** The object from the given scenes and noise levels was compared with the noiseless *unoccluded* scene for all five objects. The noise level is given in millimeters and the match score varies from 0 to 1 (with 1 being a perfect match). A match rank of 1 means the correct match was the first one while a rank of 5 means the recognizer thought that all other models looked more like object than the correct model. The entropy was calculated over the attempts to match against the five models.

Scene	**Noise Level**	**Score for Correct Match**	**Match Rank** (lower is better)	**Entropy** (uncertainty)
circle	0	0.767	1	2.08
circle	2	0.733	1	2.16
circle	4	0.722	1	2.13
circle	8	0.708	1	2.16
circle	16	0.703	1	2.17
circle	32	0.374	2	2.18
flyby	0	0.611	1	2.12
flyby	2	0.580	1	2.14
flyby	4	0.598	1	1.89
flyby	8	0.599	1	2.06
flyby	16	0.545	1	2.08
flyby	32	0.256	3	2.16
unoccluded	0	1.000	1	1.95
unoccluded	2	0.770	1	2.09
unoccluded	4	0.737	1	2.11
unoccluded	8	0.736	1	2.11
unoccluded	16	0.677	1	2.01
unoccluded	32	0.269	2	2.16
Recognition rate: 83%				

Table 6.4: **Recognition results for *semi*.** The object from the given scenes and noise levels was compared with the noiseless *unoccluded* scene for all five objects. The noise level is given in millimeters and the match score varies from 0 to 1 (with 1 being a perfect match). A match rank of 1 means the correct match was the first one while a rank of 5 means the recognizer thought that all other models looked more like object than the correct model. The entropy was calculated over the attempts to match against the five models.

Scene	Noise Level	Score for Correct Match	Match Rank (lower is better)	Entropy (uncertainty)
circle	0	0.603	2	2.26
circle	2	0.628	1	1.93
circle	4	0.682	1	1.92
circle	8	0.636	1	1.93
circle	16	0.588	1	1.95
circle	32	0.511	2	2.26
flyby	0	0.444	1	2.30
flyby	2	0.187	5	2.29
flyby	4	0.430	2	2.29
flyby	8	0.234	5	2.30
flyby	16	0.281	3	2.28
flyby	32	0.119	4	2.11
unoccluded	0	1.000	1	2.06
unoccluded	2	0.826	1	2.08
unoccluded	4	0.831	1	2.06
unoccluded	8	0.473	2	2.11
unoccluded	16	0.697	1	2.26
unoccluded	32	0.237	4	2.13
Recognition rate: 50%				

Table 6.5: **Recognition results for *tank*.** The object from the given scenes and noise levels was compared with the noiseless *unoccluded* scene for all five objects. The noise level is given in millimeters and the match score varies from 0 to 1 (with 1 being a perfect match). A match rank of 1 means the correct match was the first one while a rank of 5 means the recognizer thought that all other models looked more like object than the correct model. The entropy was calculated over the attempts to match against the five models.

CHAPTER 7

CONTRIBUTIONS AND FUTURE WORK

7.1 Summary and Conclusions

We have developed a system which takes synthetic range data from a hypothetical unmanned vehicle and recognizes the vehicle being imaged. The range data is cluttered by a pessimistic simulation of a leaf canopy and corrupted by isotropic Gaussian noise. Taking this data, we use local surface fits to initially filter some of the clutter and reduce our data size prior to reconstructing a mesh surface. This mesh surface then has spurious triangles removed and is segmented using a method based on the normalized cut of an affinity matrix constructed from the local surface fits. These segments are then matched to those in a modelbase using a tree-based inexact partial graph matcher.

For two of our five objects, we achieve 100% correct recognition across all three scenes and all six noise levels. In the cases where our recognition engine fails to properly identify the object, the leading cause is over-segmentation and the second most frequent cause is due to portions of the surface not being seen that lead to errors in matching segment areas.

7.2 Improvements and Future Work

There are a number of ways in which the system we present in this thesis could be improved and/or extended:

1. **Articulation:** Currently we do not consider cases of articulation such as a rotating gun turret or a semi truck with a trailer being towed. Because many military vehicles are articulated, a final system would need to be able to handle this situation. By properly adding articulation constraints to the models, we believe that our approach could be extended to handle articulation.

2. **Larger modelbase:** Our experiments were done on five different models. A deployed system would need to accurately and quickly identify from a model base of at least hundreds or thousands of objects.

3. **Iterative recognition:** In a hostile environment, it is important to properly budget sensor time so as to not put the sensor platform at undue risk as well as to avoid alerting any potential targets that they have been identified. An iterative recognition loop that takes the range images already produced and hypothesizes a match can then decide whether more data are required to reduce the uncertainty in the match. If more data are required, a view planning algorithm could be employed to set the optimal flight path to acquire the needed data.

4. **Alternative Segmentation Methods:** Although we were able to work with several methods for segmenting our meshes, there were many others which we

did not have an opportunity to explore to see if they could better overcome the over-segmentation problem. Some alternatives are:

(a) the different sparse matrix normalization scheme proposed by Ng *et al.* [20].

(b) tensor voting methods such as those proposed by Tang and Medioni [29]

(c) using the full segmentation algorithm for the robust sequential estimator, as detailed in [3].

(d) heuristically enhancing Srikantiah's segmentation algorithm [26] to be able to detect orientation conflicts when growing segments.

5. **Verification:** Especially as the modelbase size grows, a verification step may be necessary to evaluate the most likely matches. A popular method of verification is to attempt to register the object to its model using the segment matches to initialize the registration.

6. **Alternative Recognizers:** Although we are suspicious about the ability of appearance based recognition techniques to deal well with the clutter and noise in our scenes, a recognizer based on spherical attribute images might be made to work in this environment.

APPENDIX A

SOFTWARE AND TIMING INFORMATION

A.1 Software and Libraries Used

We developed our source code using Microsoft Visual C++ 6.0 on MS Windows and using KDevelop and GNU C++ on Linux. We used the following 3rd-party libraries and development tools as well:

- **Matlab** for system prototyping, especially in evaluating some of the alternative spectral segmentation techniques [37].
- **Normalized Cuts** for the segmentation algorithm we used. Unfortunately, the implementation we downloaded from Berkeley is no longer available from them.
- **Perl** for scripting and automating portions of our system for execution and data analysis [36].
- **POV-Ray**, a ray tracing program which we modified to output range images [39].
- **Ravi Srikantiah's Source** for his Master's project served as an initial starting point for our work [26].

Step	**Machine Specifications**	**# of Times Performed**	**Total Time (hh:mm)**
Range image generation	Pentium II, 450MHz SuSE Linux 7.3	780	7:34
Local surface fitting	Pentium IV, 1.7GHz MS Windows XP	90	0:57
Surface reconstruction	Pentium IV, 1.7GHz MS Windows XP	90	2:14
Segmentation	Pentium III, 1GHz MS Windows 2000	90	1:07
Recognition	Pentium III, 1GHz MS Windows 2000	90	0:02

Table A.1: **Approximate execution times.** The number of range images generated is the number of scenes times the number of objects times the number of camera positions. The number of times each of the other steps was performed is the number of objects times the number of scenes times the number of noise levels.

- **VTK** (Visualization Toolkit) for most of renderings [23], [24], [40].
- **VxL** for numerical computations, especially for performing PCA and the biquadratic fits [41].

A.2 Execution Times

We give estimates of the execution times required for the major steps in our recognition procedure in Table A.1.

BIBLIOGRAPHY

[1] Nina Amenta and Marshall W. Bern. Surface reconstruction by voronoi filtering. In *Symposium on Computational Geometry*, pages 39–48, 1998.

[2] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *International Journal on Computational Geometry and Applications*, 12:125–141, 2002.

[3] Kim L. Boyer, Mohammad J. Mirza, and G. Ganguly. The robust sequential estimator: a general approach and its application to surface organization in range data. *Pattern Analysis and Machine Intelligence*, 16(10):987–1001, Oct. 1994.

[4] Richard J. Campbell and Patrick J. Flynn. A survey of free-form object representation and recognition techniques. *Computer VIsion and Image Understanding*, 81(2):166–210, February 2001.

[5] C. H. Chen and A. C. Kak. A robot vision system for recognizing 3-d objects in low-order polynomial time. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1535–1563, November/December 1989.

[6] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Proceedings of SIGGRAPH*, pages 303–312, August 1996.

[7] Tamal K. Dey and J. Giesen. Detecting boundaries for surface reconstruction using co-cones. *Proceedings of the International Journal of Computer Graphics & CAD/CAM*, 16:141–159, 2001.

[8] Tamal K. Dey and P. Kumar. A simple provable algorithm for curve reconstruction. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algoritms (SODA '99)*, pages 893–894, 1999.

[9] T. K. Dey, J. Giesen, S. Goswami, and W. Zhao. Shape dimension and approximation from samples. In *Proceedings of the 13th ACM-SIAM Symposium Discrete Algorithms*, pages 772–780, 2002.

[10] Patrick J. Flynn and Anil K. Jain. Surface classification: Hypothesis testing and parameter estimation. In *Proceedings of Computer Vision and Pattern Recogntion*, pages 261–267. IEEE, IEEE Press, 1988.

[11] Patrick J. Flynn and Anil K. Jain. On reliable curvature estimation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 110–116, 1989.

[12] Patrick J. Flynn and Anil K. Jain. CAD-based computer vision: from CAD models to relational graphs. *Pattern Analysis and Machine Intelligence*, 13(2):114–132, February 1991.

[13] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Company, 2nd edition, 1992.

[14] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, 2nd edition, 1989.

[15] Martial Hebert, Katsushi Ikeuchi, and Hervé Delingette. A spherical representation for recognition of free-form surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):681–690, July 1995.

[16] Hughes Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Proceedings of SIGGRAPH*, pages 71–78, 1992.

[17] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. In *Foundation of Computer Science*, number 41, pages 367–377. IEEE, 2000.

[18] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM, ACM Press, 1987.

[19] M. J. Mirza and K. L. Boyer. A robust sequential procedure for surface parameter estimation and curvature computation. In *International Conference on Intelligent Robots and Systems*, pages 2017–2026. IEEE/RSJ, July 1992.

[20] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Neural Information Processing Systems*, number 14, page To appear, December 2002.

[21] P. Perona and W. T. Freeman. A factorization approach to grouping. In *European Conference on Computer Vision*, pages 665–670, June 1998.

[22] Sudeep Sarkar and Kim L. Boyer. Quantitative measures of change based on feature organization: Eigenvalues and eigenvectors. *Computer Vision and Image Understanding*, 71(1):110–136, July 1998.

[23] W. J. Schroeder and K. M. Martin. *The VTK User's Guide*. Kitware, Inc., June 1999.

[24] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall PTR, 2nd edition, 1998.

[25] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[26] Ravi Srikantiah. Multi-scale surface segmentation and description for free form object recognition. Masters thesis, The Ohio State University, 2000.

[27] Henry Stark and John W. Woods. *Probability, Random, Processes, and Estimation Theory for Engineers*. Prentice-Hall, Inc., 2nd edition, 1994.

[28] Gilbert Strang. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich College Publishers, 3 edition, 1988.

[29] Chi-Keung Tang and Gérard Medioni. Curvature-augmented tensor voting for shape inference from noisy 3d data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6):858–864, June 2002.

[30] Song Wang and Jeffrey Mark Siskind. Image segmentation with ratio cut. *Submitted to Pattern Analysis and Machine Intelligence*, 2002.

[31] Alan H. Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley Publishing Company, 1st edition, 1992.

[32] Y. Weiss. Segmentation using eigenvectors: a unifying view. In *International Conference on Computer Vision*, pages 975–982. IEEE, 1999.

[33] Andrew K. C. Wong, Si W. Lu, and Marc Rioux. Recognition and shape synthesis of 3-d objects based on attributed hypergraphs. *Pattern Analysis and Machine Intelligence*, 11(3):279–290, March 1989.

[34] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner, and OpenGL Architecture Review Board. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Vesion 1.2*. Addison-Wesley Pub Co, 3 edition, 1999.

[35] Yizhou Yu, Andras Ferencz, and Jitendra Malik. Extracting objects from range and radiance images. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):351–364, Oct.–Dec. 2001.

[36] *Comprehensive Perl Archive Network*. `http://www.cpan.org`.

[37] *The MathWorks Homepage*. `http://www.mathworks.com`.

[38] *Netlib Repository*. `http://www.netlib.org`.

[39] *Persistence of Vision Raytracer*. `http://www.povray.org`.

[40] *The Visualization Toolkit*. `http://www.kitware.com`.

[41] *The VxL Homepage*. `http://vxl.sourceforge.net`.