A Consensus Approach to Primitive Extraction

Gerhard Roth

B. Math. (Hons), (University of Waterloo), 1976
M. Comp. Sci., (Carleton University), 1984

Department of Electrical Engineering
McGill University
Montréal

May, 1993

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

ISBN 0-315-87904-1

Canada

Abstract

This thesis applies the consensus paradigm to an important problem in model-based vision, that of primitive extraction. Primitive extraction is the process of finding geometric primitives in geometric data. Such data are obtained directly by active sensors such as laser rangefinders, by processes that operate on passive sensor data to create depth information such as stereo vision, or by simple edge detection and thresholding of intensity images. A geometric primitive is a curve or surface which can be described by an implicit function. We show that the best solution to this problem is at the global optimum of a cost function which often has very many local optima.

This global optimum represents the best consensus in the data with regard to the extraction problem. The consensus paradigm attempts to find this global optimum by randomly choosing small subsets of the data and evaluating the cost function for each subset. We apply the consensus paradigm to this problem by randomly sampling minimal subsets. This is a simple and general way of finding the best consensus. For primitive extraction a minimal subset is the smallest number of points necessary to define a geometric primitive. The issues of how to choose the appropriate cost function, how to decide on the number of random samples, and how to convert between a minimal subset and the parameter vector that defines the primitive are explored in detail.

While effective, the consensus approach using random sampling often requires a large number of random samples, and therefore a large number of cost function evaluations. We address this problem by combining the consensus paradigm with a genetic algorithm that uses the minimal subset representation. A genetic algorithm is an optimization method based on the evolutionary metaphor. It has been successfully applied to difficult optimization problems, where the cost function is noisy, multi-dimensional, and has many local minima. In our applications the genetic algorithm is able to use local geometric information to produce a global solution in a way that

usually avoids the problem of premature commitment. The resulting method often requires far fewer cost function evaluations than the random sampling approach. Some ways of implementing these algorithms on different parallel architectures are described.

Résumé

Cette thèse applique le paradigme du consensus à un problème important en vision basée sur des modèles: l'extraction de primitives. L'extraction de primitives est le processus de repérage de primitives géométriques dans des données géométriques. Ces données proviennent directement de capteurs actifs tels que les télémètres au laser, de processus opérant sur des données de capteurs passifs pour en extraire l'information de profondeur tel que la vision stéréo, ou par simple détection d'arêtes et seuillage d'images d'intensité. Une primitive géométrique est une courbe ou une surface qui peut être décrite par une fonction implicite. Nous démontrons que la meilleure solution à ce problème se trouve à l'optimum global d'une fonction de coût qui possède souvent de très nombreux optimums locaux.

Cet optimum global représente le meilleur consensus obtenu des données en ce qui concerne le problème d'extraction. Selon le paradigme du consensus, on cherche à trouver cet optimum global en choisissant de façon aléatoire de petits sous-ensembles des données et en évaluant la fonction de coût pour chaque sous-ensemble. Nous appliquons le paradigme du consensus à ce problème en choisissant au hasard des sous-ensembles minimaux. Il s'agit d'une façon simple et générale de trouver le meilleur consensus possible. Pour l'extraction de primitives, un sous-ensemble minimal est le nombre minimum de points définissant une primitive géométrique. Nous discuterons du choix de la fonction de coût, de la détermination du nombre d'échantillons aléatoires, et de la conversion entre un sous-ensemble minimal et un vecteur de paramètres définissant la primitive.

Bien qu'efficace, l'approche par consensus utilisant l' échantillonnage aléatoire nécessite souvent un grand nombre d'échantillons aléatoires, et par conséquent un grand nombre d'évaluations de la fonction de coût. Nous abordons ce problème en combinant le paradigme de consensus avec un algorithme génétique qui utilise la représentation par sous-ensembles minimaux. Un algorithme génétique est une méthode d'optimisation fondée sur la métaphore de l'évolution. Cette méthode a été

appliquée avec succès à des problèmes difficiles d'optimisation, où la fonction de coût est bruitée, multi-dimensionnelle et possède plusieurs minimums locaux. Dans nos applications, l'algorithme génétique est en mesure d'utiliser l'information géométrique locale afin de produire une solution globale en évitant habituellement le problème de décision prématurée. La méthode qui en découle requiert souvent beaucoup moins d'évaluations de la fonction de coût que la méthode d'échantillonnage aléatoire. Nous décrirons quelques méthodes de réalisation de ces algorithmes sur diverses architectures parallèles.

Acknowledgements

First I would like to thank my supervisor, Dr. Martin D. Levine for many years of encouragement and direction. He supported my efforts over the difficult years when it seemed that I would never finish.

There are many students at McGill who made my years there more stimulating. In particular Allan Dobbins, Lee Iverson, Benjamin Kimia, Guy Godin, Wade Hong and Robert Bergevin come to mind. Special mention must be given to Robert Bergevin with whom I shared an office for two years. He not only gave me advice but became a good friend.

Thanks to my wife Cathy who has put up with a husband that has returned to graduate school not once, but twice. She has supported me faithfully without complaint. My children Andrew, Heather, and Fiona have made me look forward to being home more often. Also thanks to my parents, Emmy and Bill, who always encouraged me to reach my potential, especially academically.

Many colleagues at the National Research Council of Canada have helped me in my work. They are too numerous to mention, but particular thanks go to Colin Archibald, Jagdeep Basran, Michael Greenspan, and Guy Godin. All have listened to me for many hours, and Guy especially has been most patient.

Last but not least, my employer, the National Research Council of Canada, provided education which made my stay at McGill possible. My supervisors, Nestor Burtnyk approved the leave, and Jacques Domey supported me on completing my degree. Mr. Alex Mayman, the Director General of the Institute for Information Technology was instrumental in having the leave approved.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

We believe that any model-based vision system will require the ability to extract geometric primitives. This means finding geometric primitives (such as lines, planes, spheres, etc.) in geometric sensor data. Once this basic capability exists it can be used to build particular solutions to such model-based vision problems as segmentation [Roth and Levine, 1990b], pose determination and pose refinement [Roth and Levine, 1991b].

An example of primitive extraction is shown in Figure 1.1 where two ellipses are extracted from an image of a cable surrounded by an insulator. Part (a) shows the initial image, with part (b) indicating a set of edge points computed from this image. Part (c) presents the two extracted ellipses drawn as dark lines. The fact that the two ellipses are very close together makes the extraction difficult. This example will be discussed in more detail later in the thesis.

Our solution to the extraction problem is based on the consensus paradigm [Meer, 1991], which Meer defines as follows:

> "Compute a candidate model based on a randomly chosen small subset of the data. Apply this model to all the data. Compute a global measure for the model. Optimize the quality measure by repeating the procedure several times."

This paradigm embodies the concept of feedback [Besl and Jain, 1985], in which each potential solution is tested against the original geometric data. The first application in the computer vision field of the consensus approach was the RANSAC algorithm which was used to register images [Fischler and Bolles, 1981, Bolles and Fischler, 1981]. It was then rediscovered in the statistics field in the form of the Least Median Squared (LMedS) algorithm [Rousseeuw and Leroy, 1987], and used as the basis of a number of segmentation algorithms [Kim *et al.*, 1989, Meer *et al.*, 1990, Roth and Levine, 1990b]. In this thesis we apply the consensus paradigm to the

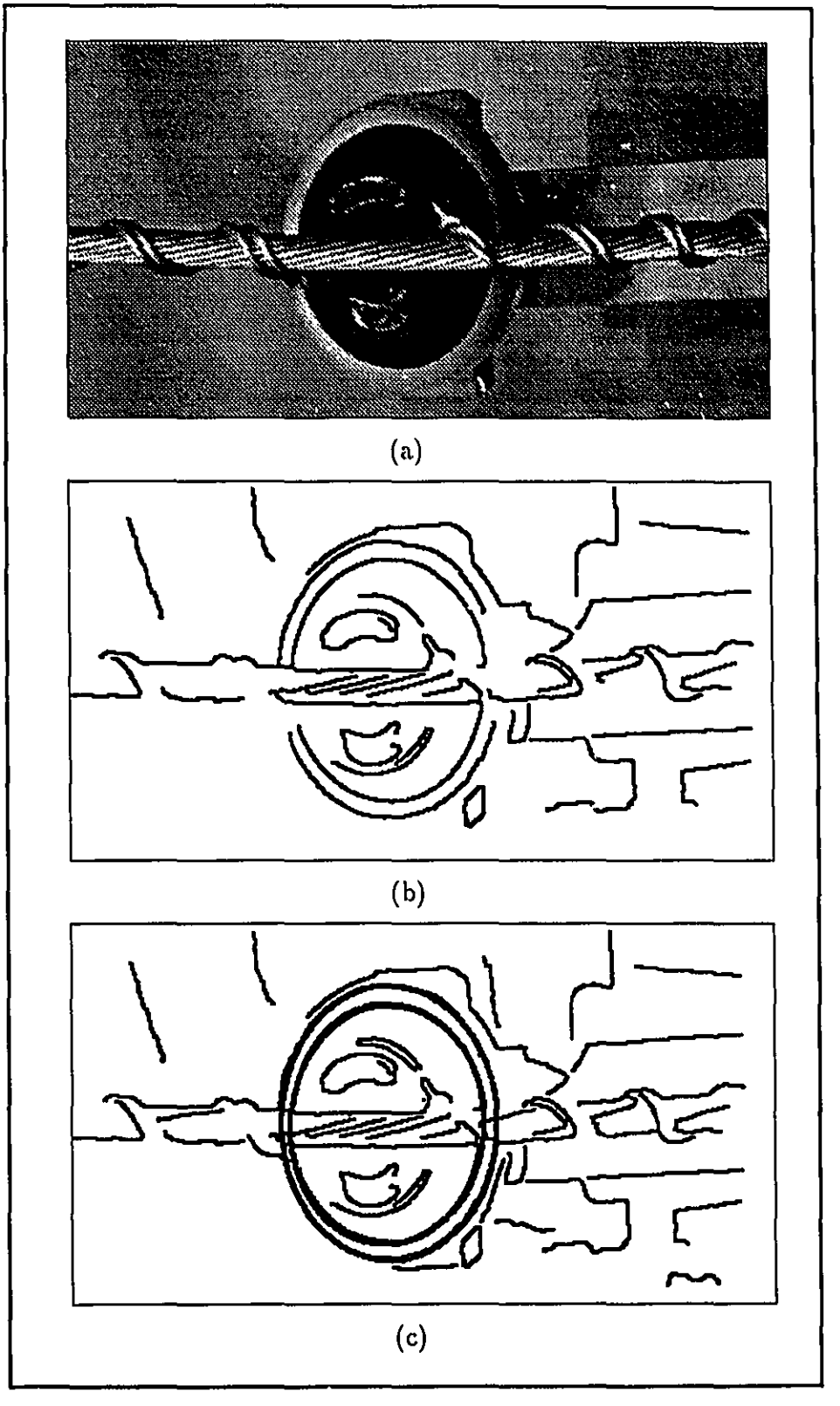

Figure 1.1: Extracting ellipses from cable image (a) initial intensity image (b) edge point chains (c) two extracted ellipses

problem of primitive extraction by performing random sampling of minimal subsets. We also combine the consensus approach with a genetic algorithm (GA). A GA is an optimization procedure which is based on an analogy with evolution. We believe that the resulting solutions to the extraction problem have a number of unique capabilities and characteristics.

1.1 Motivation

Before we define this problem more precisely, we will first explain why it is important. One thing that has become clear in the last twenty years is that if we wish to have operational computer vision systems there must be constraints on the environment. Without such constraints it is difficult, if not impossible, to produce clear problem definitions, and to evaluate potential solutions. In the computer vision field the most common constraint is that of model-based vision. In this paradigm the simplifying assumption is made that the environment consists of manufactured objects whose geometry is known beforehand. This is a reasonable assumption given the fact that most manufactured objects are created with the aid of a computer-aided design (CAD) package [Faux and Pratt, 1979].

To reiterate, in model-based vision the assumption is that there exists a model of every significant object in the environment. Each of these objects is defined as a set of geometric primitives. A geometric primitive is a curve or surface which is described by an equation with a number of free parameters. One example of a model would be the description of a room as a set of line segments, another the description of a cube as a set of planar patches. More complex models have a wider variety of defining geometric primitives, but the basic principle is unchanged.

The next essential component in a model-based vision system is sensor data. Here, we would like to be as general as possible in describing how this data is presented. We feel that this can best be done by thinking of the sensors as providing an unordered list of points in two-or three-dimensional Cartesian space, which we call geometric data. The fact that the set of points are unordered is important for a number of reasons. The

first reason is that the process that created the sensor data may not produce any such ordering. The second reason is that the data may be taken from a number of different viewpoints. In this case it is still an open research question as to how to produce an ordering among the data points [Aubry and Hayward, 1988, Hoppe *et al.*, 1992]. However, in situations where an ordering between geometric data points does exist we will show how it can be used advantageously in our algorithm.

In model-based vision there are three basic tasks that appear repeatedly: object identification, pose determination and pose refinement. Object identification is the process of identifying which objects in the sensed environment are objects in the CAD database. Pose determination assumes that an object has been so identified, and that we want to find its pose. The pose of an object is its position and orientation, and this information is essential for manipulation and planning purposes. Pose refinement is the process of improving or refining this pose, and is essential for tracking applications. The assumption here is that the object has been identified and its approximate pose has already been determined. There are many different approaches to these three problems, and they vary depending on the particular application.

Are there any commonalties that link these three problems? That is, is there any basic capability upon which all solutions to these problems and other model-based vision problems depend? We believe that there is, and that this basic building block is the extraction of geometric primitives. This is the process of finding primitives (such as lines, planes, spheres, etc.) in geometric sensor data. The problems of object identification, pose determination and pose refinement all require primitive extraction. Therefore by focusing on the problems of primitive extraction we are not providing solutions to any particular model-based vision problem. Instead, we are creating a tool which can be used to build solutions to a wide variety of tasks in model-based vision.

1.2 Overview

In order to better understand the problem of primitive extraction we define an optimization model for this task. The use of such models has recently become common in the computer vision field [Leclerc, 1989]. They require the creation of a cost function which measures the quality of the output produced by a particular algorithm. This cost function depends on the value of a parameter vector, and the goal is to find the parameter vector value which optimizes the cost function. For primitive extraction the cost function measures the quality of the extracted primitive. In the case of extraction the parameter vector defines the primitive. The best value for the parameter vector is the one for which the cost function is at its global optimum. This solution represents the best consensus in the data according to the criteria encoded in the cost function.

This optimization model for extraction gives us a number of important insights. The first is that this problem does indeed fit into an optimization framework. The second, is that algorithms for this task must attempt to find the global optimum of cost functions which have many local optima. Each local optimum partitions the input into two groups, which we call inliers and outliers. For primitive extraction inliers are geometric data points that belong to the primitive, and outliers are the remaining geometric data points.

It is important that any algorithm for solving the problems of primitive extraction be robust. In an intuitive sense robustness is the ability to achieve a good solution even in the case where there are many outliers. The quality of the solution depends on how close the optimum value of the cost function found by the algorithm is to the global optimum. We show how various algorithms for primitive extraction (such as the Hough transform), can be recast in this optimization framework by simply defining the appropriate cost function. This optimization model indicates that all robust algorithms have the goal of finding the global optimum from among many local optima.

The number of local optima is potentially as large as the number of ways to

partition the input into inliers and outliers. Since this is a very large number, primitive extraction is in many cases computationally difficult. While these problems are in theory solved once a cost function has been defined, an *efficient* solution must find a value close to the global optimum in as few evaluations of the cost function as possible.

The minimal subset principle states that a subset of a set can often encode the characteristics of the entire set. For primitive extraction a minimal subset is the smallest number of points necessary to define a geometric primitive. For example, in the case of a line the minimal subset contains two points, for a plane three points, etc. We show that for accurate geometric data efficient solutions to the problem of primitive extraction can be obtained using minimal subsets.

One way to solve the problem of primitive extraction is to evaluate the cost function only at the values of the parameter vector defined by the minimal subsets. If the geometric data are without error, then this procedure is guaranteed to find the true global optimum. As the accuracy of the geometric data decreases, the likelihood of this occurring also decreases. While the requirement for very accurate geometric data may seem stringent, it is the case that modern sensors usually provide such data.

Evaluating the cost function at all possible minimal subsets is rarely practical. A more efficient way is to evaluate the cost function only at the parameter vector values defined by randomly chosen minimal subsets. We show that this approach, which we call minimal subset random sampling, is a simple and general way to solve the problem of primitive extraction. We discuss a number of important issues in the implementation of this algorithm, including how to map between a minimal subset and a parameter vector. We show how such a mapping can be created efficiently for a wide variety of geometric primitives by using elimination theory, which is a method from the field of symbolic algebra.

While effective, minimal subset random sampling still may require too many cost function evaluations. To overcome this problem we combine the minimal subset approach with a genetic algorithm (GA). A GA is a procedure which has been shown to efficiently find an optimum value which is often close to the global optimum for

noisy cost functions which have many local optima. It is based on an analogy with evolution, in which the new solutions are created from a population of potential solutions. In our case a population member defines the parameter vector for the extraction problem, and is represented by a minimal subset. We apply a GA to the problem of primitive extraction. The initial population is created by random sampling over small components of the geometric data. The GA takes this initial population and uses genetic operators to create better solutions. It takes advantage of the local information provided by the initial population to quickly converge to a global solution. The resulting extraction algorithm often requires substantially fewer cost function evaluations than a method which uses only random sampling.

Real-time performance is often necessary in model-based vision applications. For this reason we show how these algorithms can be implemented on parallel hardware. We demonstrate that this can be done on a wide variety of parallel architectures, which is a significant advantage of our approach over other extraction algorithms.

This research has produced a better understanding of the inherent complexity of the problems of primitive extraction. Our solutions to this problem are already practical for many applications, and will become more so as parallel hardware becomes more widely available. We believe that the use of such consensus algorithms will lead to significant advances in the field of model-based vision because of their robust nature.

1.3 Definitions

Since the terms geometric data and geometric primitives are essential to the understanding of this thesis we will explain their meaning in more detail. We will begin with the idea of geometric data. This concept is a slightly modified version of the term geometric signal [Besl, 1990]. Geometric data are nothing more than an unordered list of points in two-or three-dimensional Cartesian space. Such data are obtained from two basic methods. The first is to use a passive sensor (such as an ordinary intensity camera) followed by processes such as edge detection or stereo vision. The second is

to use an active sensor (such as a laser rangefinder) to probe the environment. For our algorithms neither the method used to produce the data nor the dimensionality is important.

There is a number of reasons why we emphasize the fact that no ordering is assumed for the geometric data. The first reason is that the process that produces the geometric data may not create such an ordering. An example of this is the matching process of stereo vision. Another example is when the sensor itself does not produce data on a regular grid [Blais *et al.*, 1991]. The second reason is that sometimes the data may be taken from a number of different views. In this case there is no simple way to find the ordering of the points between the different views (see [Aubry and Hayward, 1988] for research in this area). The third reason is that we believe that the lack of ordering makes it much easier for our algorithms to be implemented on parallel hardware.

We also argue that it is often the case that the number of geometric data points is limited; that is, in the order of thousands of points. Certainly for geometric data extracted from intensity images by processes such as edge detection or stereo this is true. It is not the case for dense geometric data (sometimes called range data) obtained using an active sensor. We believe that dense range data is unnecessary for many robot vision tasks, and that relatively sparse range data (thousands versus millions of points) is more than sufficient. However, for certain inspection tasks dense data is appropriate. For these reasons we have made our definition of geometric data general enough to accommodate both sparse and dense geometric data.

We will now define more precisely what we mean by a geometric primitive. In model-based vision it is assumed that there is a geometric description of each significant object in the environment. Each such object is described as the union of geometric primitives. A geometric primitive is a curve or surface which is defined by an equation with a number of free parameters. Examples of such primitives in 2D are lines, circles, and ellipses, and in 3D are planes, spheres, tori. If the object were a cube it could be described as the union of six planar patches. More complex objects need more complex primitives to describe them, but the principle is the same.

A geometric primitive can be defined in two different ways: using the parametric form or the implicit form. The parametric form is commonly used in computer graphics and computer-aided design systems. It is most useful for generating the points on a curve or surface, or for manipulating the curve or surface. By contrast, the implicit form is most useful for finding whether a given point is on a particular curve or surface. A curve or surface in implicit form is defined as the set of points which are the zeros of a function. In this thesis the notation used to describe a geometric primitive in implicit form is $f(\overline{p};\overline{a}) = 0$. In this notation, $\overline{p}$ is the datum point, and $\overline{a}$ defines the parameter vector for this particular primitive. For example, a 2D line is defined implicitly by the equation $a_0 + a_1 x + a_2 y = 0$, where the parameter vector $\overline{a}$ is (a_0, a_1, a_2) and the datum point $\overline{p}$ is (x, y). Different instances of the geometric primitive are produced by changing the parameter vector $\overline{a}$. The implicit form naturally divides space into three regions, $f > 0$, the points on one side of the curve or surface, $f < 0$, the points on the other side of the curve or surface, and $f = 0$ the points on the curve or surface. The implicit form is used in this thesis when describing geometric primitives. The reason is that for primitive extraction it is necessary to efficiently compute the closest distance of a point to a curve or surface. This can be with the implicit form using an approximation that is adequate for this application. Since the object models in CAD databases are described in parametric form, it is necessary to be able to convert from the parametric to implicit form in order to use our algorithms. This can be done by using elimination theory, which is a method from the field of symbolic algebra.

1.4 Related Work

The RANSAC algorithm was the first to use the consensus approach in the computer vision field [Fischler and Bolles, 1981, Bolles and Fischler, 1981]. The consensus approach has also been used in the robust statistics field [Rousseeuw and Leroy, 1987]. Recently this method has been more widely applied [Meer *et al.*, 1991, Jolion *et al.*, 1991, Mintz, 1991]. Some of this research uses con-

sensus methods to solve the segmentation problem [Jolion *et al.*, 1991], other work attacks a variety of computer vision problems [Mintz, 1991]. We concentrate only on the problem of primitive extraction. Other robust statistics methods have also been applied to problems such as local estimation of surface properties [Besl *et al.*, 1988]. Such research adheres closely to the terminology of the robust statistics approach, especially in the understanding of breakdown. Our work differs in the understanding we have of this concept, whose utility in the computer vision field we consider questionable.

Recently, some work from the field of computational geometry has been done to find efficient algorithms for robust estimation [Dillencourt *et al.*, 1992, Netanyahu, 1992]. These algorithms find the exact solutions to certain robust estimators without using random sampling. However, the expected running time is only achieved with a given probability, and they may execute very slowly. While these methods hold promise they have been shown to work only for robust estimators that extract lines, and it is not clear how to extend them to deal with more complex primitives.

The Hough transform has had a long history of both theoretical and practical results in the computer vision field [Illingworth and Kittler, 1988]. It has been widely applied to the problem of primitive extraction. While our work considers the same problem it does so in a completely different fashion. The Hough transform uses a parameter space to accumulate votes for different geometric primitives. Once the voting is finished the parameter space is analysed to find the best set of primitives. By contrast we do not build a parameter space, but evaluate each hypothesized geometric primitive directly by comparing it against the geometric data. Our work is thus more similar to the hypothesize-and-test methodology than the Hough transform. While this evaluation process is a potential computational bottleneck, we believe that it can be easily parallelized using parallel hardware. Recently there has been some research on overcoming some of the deficiencies of the Hough transform by creating randomized versions [Oja and Xu, 1990, Kiryati *et al.*, 1991, Bergen and Shvaytser, 1991]. While this approach is similar to ours in that random sampling is used, a voting

process in parameter space still occurs. This is not true for our method, which does not use a voting process, but instead evaluates each hypothesized geometric primitive directly against the geometric data.

Other recent work explores the use of implicit functions in model-based computer vision [Taubin, 1988, Taubin and Cooper, 1990, Taubin, 1991, Bolle and Vemuri, 1991, Kriegman and Ponce, 1990, Ponce *et al.*, 1991]. The relationship between parametric and implicit descriptions of curves and surfaces has already been noted and explored [Bolle and Vemuri, 1991]. The implicit form can be obtained by conversion from the parametric form [Sederberg and Anderson, 1984], which is how objects are described in conventional CAD data bases. Given some geometric data, and a curve or surface in implicit form, the best fit to the geometric data points can be determined. This fitting process is not a simple task [Taubin, 1991, Kriegman and Ponce, 1990, Ponce *et al.*, 1991], and requires complex nonlinear optimization algorithms. The key point that distinguishes our work from this body of research is that we concentrate on extraction as opposed to fitting. Extraction is a generalization of fitting which finds the best *subset* of the geometric data described by a model. By contrast, in the fitting algorithms, the assumption is made that all geometric data points belong to a single primitive.

The area of range image segmentation is a rapidly growing subfield of computer vision [Besl and Jain, 1985, Besl, 1990, Besl, 1988, Boulanger and Godin, 1992, Hoffman and Jain, 1987, Fan *et al.*, 1987]. While we also process data from laser rangefinders, our work extends beyond this category. The aforementioned papers concentrate only on the processing of dense range images. Our extraction algorithms apply to both sparse and dense range images. We process more than just range data, and besides extracting surfaces, also extract curves. We make no assumptions about the ordering of the geometric data points. In their most basic form our algorithms operate on unordered data, which is not the case for most segmentation algorithms. Extraction is not the same as segmentation, but as we will show, a repeated application of our extraction algorithm can produce a segmentation of the geometric data.

While the genetic algorithm (GA) field has become increasingly better known [Goldberg, 1988], its use in computer vision is still rare [Bhanu *et al.*, 1991, Ankenbrandt *et al.*, 1990]. Such an approach has never to our knowledge been used to solve the problem of primitive extraction. The use of a minimal subset chromosome representation in a GA is also unique. We believe that for this application this representation is superior to the traditional binary chromosome representation used in the GA literature [Holland, 1975].

1.5 Contributions

The main contribution of this thesis is robust algorithms for solving the primitive extraction problem. These algorithms are based on the minimal subset representation of the primitive. One of the solutions uses the random sampling of minimal subsets. The other uses a genetic algorithm, which is often more efficient than random sampling. The minimal subset representation is used in both cases.

The first contribution is an optimization model for primitive extraction. We show that this task is equivalent to finding the value of a parameter vector which minimizes a given cost function. We describe a variety of different solutions to this problem by changing the cost function. One implication of this optimization model is a new understanding of the robust statistics concept of breakdown. We show that this important concept is often misinterpreted, and is not completely adequate for computer vision purposes. Another implication is that a robust cost function will usually have many local minima. Thus there is an intimate link between robustness and the large number of local minima of a cost function. In order to be successful, an extraction algorithm must be able to find a value close to the global optimum of the cost function from among the many local optima. For reasons of efficiency this should be done with as few evaluations of the cost function as possible.

Our solution to this optimization problem uses the concept of minimal subsets. We show that for perfectly accurate geometric data, random sampling of minimal subsets is probabilistically guaranteed to find the global optimum of the cost function. For

highly accurate geometric data the optimum found by random sampling will still be close to the global optimum. We demonstrate that minimal subset random sampling is a simple approach to solving the general problem of primitive extraction. We study how to choose the number of random samples, how to select a cost function and efficiently evaluate it, and how to map from a minimal subset to a parameter vector. This mapping is necessary since evaluating the cost function requires the parameter vector of the curve or surface. We investigate how an efficient closed form implementation of this mapping can be obtained by using elimination theory from the field of symbolic algebra. We show that this can be done for a wide variety of different geometric primitives.

This means that in theory our approach could be used to build a general extraction system which takes the equation of a geometric primitive to be extracted, and produces a random sampling and genetic algorithm for extracting this primitive. The reason for this generality is that the equation that maps from a minimal subset to a parameter vector is the only part of the extraction algorithm that changes with the geometric primitive. We demonstrate this in the thesis by using the same basic algorithm for extracting a wide variety of different geometric primitives. In our examples the actual program that performs this mapping is created manually, and inserted in the extraction algorithm at the appropriate place. However, because we create this mapping equation in closed form using a symbolic method, it is possible in theory to automate this procedure to generate the entire extraction algorithm directly from the equation of the geometric primitive to be extracted.

The random sampling approach sometimes requires too many samples for success. To decrease the number of random samples, and therefore the number of cost function evaluations, we apply a genetic algorithm (GA) to the extraction problem. Our implementation of the GA is unique in that it uses a minimal subset chromosome representation. A GA requires an initial population of potential solutions, and this is created by a variety of different approaches. The GA operates on this initial population by applying genetic operators to improve the quality of the solutions. The resulting algorithm is often substantially more efficient than a purely random

sampling approach.

The ability to implement our solutions on parallel hardware is essential for real-time performance, or when a large amount of geometric data is involved. By parallelizing the evaluation of the cost function it is possible to achieve substantial speedups. These speedups are obtained for both the random sampling and genetic algorithm version of our extraction algorithm. This part of the algorithm can be easily implemented on a variety of parallel hardware. We describe some possible implementations on a number of common architectures. The ease of parallelization is an important characteristic of our algorithms for primitive extraction.

1.6 Thesis Outline

Chapter Two discusses the problem of primitive extraction. We introduce the optimization model for primitive extraction and show that many previous extraction algorithms are described by this model. We then define the concept of minimal subsets, and show how the random sampling of minimal subsets solves the general problem of primitive extraction.

Chapter Three gives an overview of the field of elimination theory which is concerned with finding the symbolic solutions of a system of equations. In our algorithms it is necessary to efficiently convert between a parameter vector and a minimal subset. We show that this conversion requires the solution of a particular type of system of equations. Elimination theory, in particular the Gröbner basis algorithm, is used to achieve this goal. We demonstrate the application of elimination theory for a variety of extraction examples.

Chapter Four applies the extraction algorithm to a variety of situations. We deal with the cases where there are multiple primitives, and multiple types of primitives. We demonstrate the extraction of 2D curves, 3D curves and surfaces, and process both sparse and dense geometric data. A comparison of our approach to the Hough transform, robust fitting approaches, and the minimal length encoding schemes is made.

Chapter Five introduces the concept of a genetic algorithm (GA). Since GA's have rarely been used in computer vision, we spend some time defining the terminology of this field. The basic theoretical and practical reasons for the success of the genetic approach are discussed. We show how minimal subsets can be integrated with a GA, and how this combined approach has many advantages over a traditional GA. This hybrid approach is then applied to the problem of primitive extraction.

Chapter Six discusses various hardware-assisted speedups for these algorithms. This issue will become increasingly important as parallel architectures become more widely available. Such hardware is necessary if real-time performance is to be achieved, or if a large amount of geometric data exists. We show that by concentrating on the evaluation of the cost function our algorithms can be implemented efficiently on a variety of parallel hardware.

Chapter Seven contains the conclusions in which we expand on the contributions in order to evaluate our work in a methodical fashion. We also sketch out some further extensions to our work.

The appendices contain the solution to the various examples of elimination theory described in Chapter Three.

Chapter 2 Primitive Extraction Using Random Sampling

This chapter defines the problem of primitive extraction and presents a general solution to this problem. Primitive extraction is a generalization of fitting. The input to a fitting algorithm is some geometric data, along with a description of the primitive to be fit to this data. The output is the primitive which is the best fit to all the geometric data points. The input and output of an extraction algorithm are the same as for a fitting algorithm, with one important difference. For fitting the assumption is made that all the points truly belong to the geometric primitive. An extraction algorithm does not make this assumption. Instead, it must choose the best subset of the geometric data points described by the primitive. This set of points is called inliers, and the remaining geometric data points are called outliers.

Extraction, like fitting, can be modelled as an optimization process. In both cases the criteria used to evaluate the goodness of a geometric primitive are encoded in a cost function. This cost function measures the error between the geometric primitive and the geometric data. The best primitive is the one which optimizes the value of this cost function, thus reducing the error. The optimization model applies to both extraction and fitting. However, for the extraction problem the cost function usually has a large number of local optima. Each of these local optima represents a different division of the geometric data points into inliers and outliers. There are potentially as many local optima as there are such divisions, which is a very large number. The actual number of local optima depends on the percentage of outliers, and increases dramatically with this percentage. For fitting, on the other hand, the number of local optima is fixed. In fact, there is often only a single optimum value for a fitting problem. It is possible to change the definition of the cost function in the extraction algorithm to use different criteria for evaluating a geometric primitive. However, the fact that the cost functions normally have a large number of local optima is common to all extraction algorithms. Because of the potentially large number of local optima,

extraction is a computationally intensive task.

The goal of an extraction algorithm is to find the primitive for which the cost function achieves its global minimum. In this case the inliers represent the best consensus in the geometric data with regards to the particular type of extracted primitive. For example, if the primitive to be extracted is a line, the global optimum represents the best line in the geometric data. Of course, the meaning of best is intimately related to the cost function definition. The efficiency of any extraction algorithm depends on the number of cost function evaluations, since this is the most computationally expensive step of any extraction algorithm. An efficient extraction algorithm should find this global optimum with as few evaluations of the cost function as possible.

Our extraction algorithm is based on the random sampling of minimal subsets. A minimal subset is the smallest number of points necessary to unambiguously define a geometric primitive. For example, for a line a minimal subset contains two points, for a circle three points, etc. For accurate geometric data, a minimal subset is a good representation of all the points belonging to the primitive defined by the subset. This requirement for accurate data is reasonable given the fact that modern sensors have a very high resolution, and are thus able to produce such data. The extraction algorithm operates by repeatedly choosing a minimal subset of points randomly from the geometric data, creating a geometric primitive through these points, and evaluating the quality of the primitive using a given cost function. The primitive with the best cost function score is the one returned by the algorithm. The minimal subset random sampling extraction algorithm is independent of the cost function definition, and will successfully operate with a variety of cost functions.

For perfectly accurate geometric data this approach is guaranteed to find the global optimum if all possible random samples are taken. Since there is a large number of possible minimal subsets evaluating all of them is not practical. We show that on the average far fewer than the maximum number of such subsets need be evaluated in order to have a high confidence of success. We discuss a number of different possible cost functions, and show when it is appropriate for each to be used. A number of

simple ways to speed up the algorithm without resorting to special purpose hardware is also described.

Minimal subset random sampling has as its basis methods from the field of robust statistics (RS). This field is concerned with the situation in which the underlying statistical distribution is not Gaussian. A number of RS algorithms use random sampling, but not in the general fashion which we describe. The reasons for this are two fold. First, the robust statistics community naturally concentrates on the statistical properties of such approaches, and has less interest in providing efficient computational methods of solving the associated optimization problem. Second, the concept of breakdown used by this community is inadequate for the computer vision field. We believe that the current definition of breakdown ignores the fact that an essential component of any extraction algorithm is deciding on the significance of the results. In other words, any extraction process always returns a geometric primitive, but this primitive may or may not be significant. It is possible to use a simple statistical model to evaluate the significance of any extraction result. The breakdown concept has hindered the use of RS-based approaches by ignoring this issue, which we will discuss later in this chapter.

In the computer vision field, the Hough transform (HT) also solves the primitive extraction problem. The HT has been shown to be equivalent to a repeated template matching process [Stockman and Agrawala, 1977]. The space requirements of the HT are exponential in the number of degrees of freedom of the primitive. This restriction is intrinsic to the operation of the HT, and limits its applicability to simple geometric primitives, such as lines. The HT is not easily adapted to the extraction of more complex primitives. By contrast, the minimal subset random sampling algorithm is able to directly extract complex primitives, and is therefore more general.

While extraction is a generalization of fitting it is not equivalent to segmentation. Traditional segmentation algorithms find all the primitives at once, while our extraction algorithm only finds a single primitive at a time. This issue will be dealt with in more detail in the the next chapter. Here we only discuss the problem of extracting a single geometric primitive of a given type. We begin the chapter by

discussing fitting algorithms and their lack of robustness. Then we define the problem of primitive extraction more precisely, and discuss the HT-and RS-based solutions. We present our optimization model for fitting and extraction, and explore its most important implications. Then we introduce minimal subset random sampling, and show that this is a simple and general way of solving the extraction problem [Roth and Levine, 1990a, Roth and Levine, 1992a, Roth and Levine, In Press]. We discuss a number of important issues in the operation of the minimal subset extraction algorithm. We explore the issue of robustness and how to decide on the significance of the extraction results.

2.1 Fitting Versus Extraction

In both extraction and fitting the input consists of N geometric data points in Cartesian space along with the definition of a geometric primitive as an implicit function, $f(\overline{p};\overline{a}) = 0$. In this notation $\overline{p}$ is a datum point, and $\overline{a}$ is the parameter vector. For example, a 2D line is defined implicitly by the equation of $a_0 + a_1 x + a_2 y = 0$, where the parameter vector is (a_0, a_1, a_2), and the datum point is (x, y). The output of the fitting process is the parameter vector $\overline{a}$ of the primitive which is the best fit to the input points. In fitting, the assumption is made that all the input points truly belong to the geometric primitive. If this is not the case then the resulting fit can be arbitrarily bad. Points which belong to the primitive are called inliers, while points which do not belong are called outliers. The ability of a fitting algorithm to tolerate outliers is what we mean by robustness. A more formal definition of this concept will follow in this chapter.

Traditional fitting procedures such as least squares are not robust, as can be seen from figure 2.1. Part (a) shows a set of points in the plane, with a line fit to these points. Part (b) shows a single additional point which is not part of the line, along with the a line fit to all the points. The resulting fit has been dramatically affected by this single point, and the estimate of the line is no longer accurate. This demonstrates the lack of robustness of least squares fitting. This was the motivation for the creation

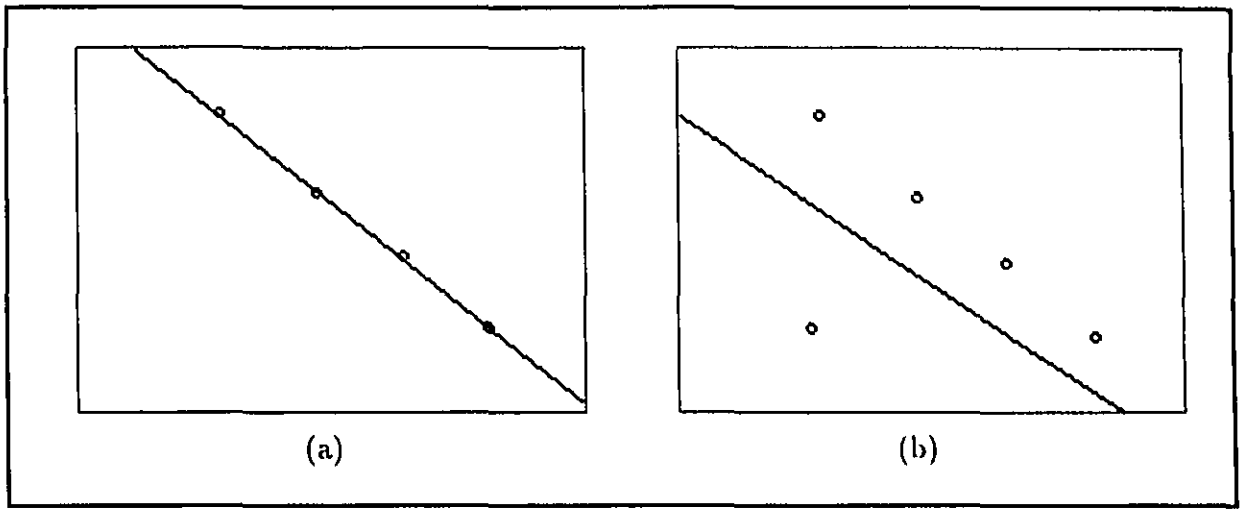

Figure 2.1: Non-robust fitting (a) Fit with all inliers (c) Fit with a single outlier.

of robust algorithms for this task.

The extraction process is identical to fitting, with one important difference. In primitive extraction the assumption is not made that all of the input points belong to the primitive. Along with the parameter vector $\overline{a}$ the extraction algorithm must choose which subset of the geometric data points are best described by the given primitive (the inliers) and ignore the rest (the outliers). The output of primitive extraction is the parameter vector $\overline{a}$ of the best primitive, along with a division of the input points into inliers and outliers. If all the input points are inliers then an extraction algorithm should produce the same results as a fitting algorithm. Thus extraction can be seen as a robust version of fitting.

This is shown clearly in figure 2.2 which shows our extraction algorithm to the same data as figure 2.1. In part (a) where there are no outliers the results are the same as the fitting algorithm. However, in part (b) the outlier has been ignored, and the correct line has been extracted. Our definition of extraction is not equivalent to robust fitting as used by the robust statistics community. This is because we do not limit the percentage of outliers to 50% as is done in the robust statistics approaches. Also our optimization model for extraction is able to describe extraction methods from both the robust statistics field (i.e. robust fitting methods) and the computer vision field (i.e. the Hough transform). Before we present this optimization model

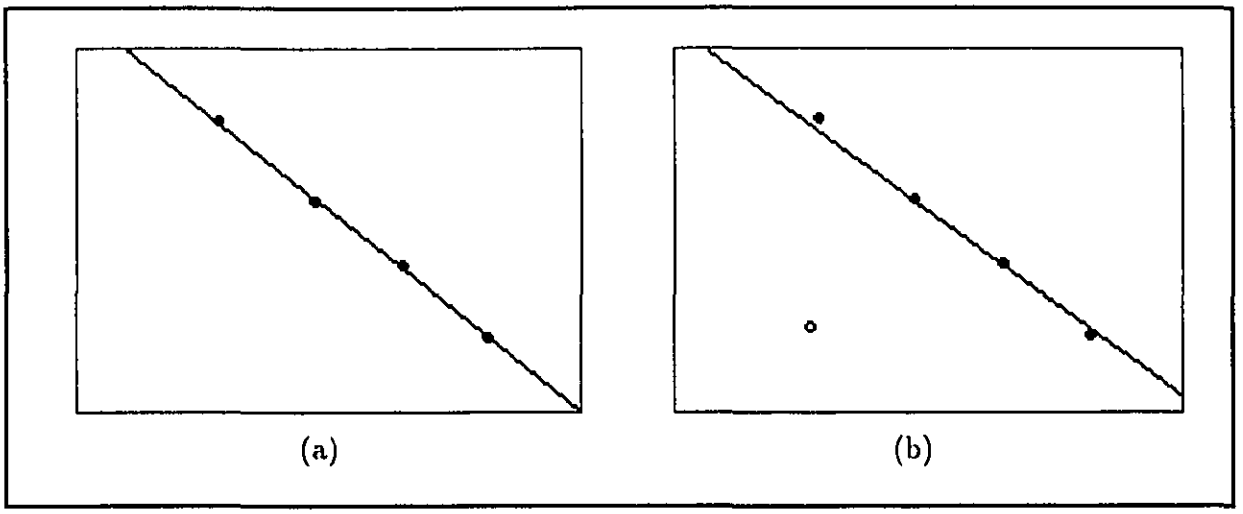

Figure 2.2: Robust extraction (a) Extraction with all inliers (c) Extraction with a single outlier.

we will discuss in more detail the HT and RS approaches to primitive extraction.

2.2 Hough Transform and Robust Fitting

In order to compare our methods to the HT and RS approaches it is necessary to have a better understanding of these two methods. For this reason we will spend some time explaining them in detail.

2.2.1 Hough Transform

The HT has had a long and varied history in the computer vision field [Illingworth and Kittler, 1988]. Its first, and still main application is the extraction of geometric primitives. The general principle of the HT is that each data point votes for all parameter combinations that could have produced it. First the parameter space is partitioned into cells (usually rectangular) by quantizing each of the dimensions of this space. Then each data point votes for every cell whose combination of parameters could have produced a geometric primitive through the given point. When all the data points have been processed, the cells which have more votes than a given threshold are selected. The parameter vector associated with each cell defines the extracted primitive, and the points that voted for each cell are the inliers.

This is the standard description of the HT that is given in computer vision textbooks [Levine, 1985]. When stated in this form the algorithm is difficult to understand because it is presented in a manner similar to the original implementation. However, it was later discovered that the HT is nothing more than a form of template matching [Stockman and Agrawala, 1977]. Since considerable insight can be gained into the operation and limitations of the HT using this viewpoint, this is the approach that we will take during our exposition.

An individual cell in parameter space describes a set of points in data space which could have been produced by any geometric primitive whose parameter vector is contained in the cell. These points taken together define a template of the same shape as the primitive. When this template is applied to the geometric data it matches the set of points which would vote for this cell in the standard HT algorithm. This makes it clear that the HT is simply a time efficient, but space inefficient way to do template matching. Assume that the templates defined by each cell of the parameter space were matched against the geometric data. Then a single point could potentially be a member of many cell templates, and would therefore be counted many separate times. By contrast, the HT processes a point only once by having it vote for all the cells for which it is a member. This is more time efficient than direct template matching, but is less space efficient, since all the cells must be represented.

The relationship of the HT to template matching makes a number of limitations of the approach clear. The first problem is that the template produced by a cell depends on the parameterization of the geometric primitive, on the cell shape and on the cell size. It was observed that different parameterizations change the template shape [Duda and Hart, 1971]. Any rectangular cell produces some distortion in the template shape relative to the ideal situation, which is to reproduce exactly the same shape as the geometric primitive. When extracting lines this distortion was lessened by using the (ρ, θ) parameterization instead of the usual slope-intercept parameterization.

It is clear that both cell shape and cell size have a direct effect on the template. Because of the distortion produced by rectangular templates, other cell shapes have been proposed, but these complicate the voting process to such a degree that they

are impractical. The template distortion lessens as the cell size decreases, which makes small cells desirable. On the other hand, the smaller the cell size the greater the number of cells, space and time requirements. Thus cell size and shape always require a compromise. Often in order to have a manageable number of cells, one is forced to use a large cell size, which in turn produces a greater distortion of the template shape, along with a decrease in the ability to isolate individual primitives. Schemes in which the quantization varies dynamically in a coarse to fine strategy have been suggested in order to decrease the storage requirements. However, this approach does not work well on complex imagery because their larger templates often contain spurious points [Princen *et al.*, 1989].

The exponential storage requirements and distortions in template shape are problems that are unavoidable when using the HT. The relation between the HT and template matching makes it clear that these limitations are intrinsic to the HT and cannot be overcome.

2.2.2 Robust Fitting

The field of robust statistics is one that has a long history [Hampel *et al.*, 1986, Huber, 1981, Gentleman, 1965], but has only recently been discovered by the computer vision community [Forstner, 1987, Besl *et al.*, 1988]. Traditional fitting algorithms assume that all the input points are inliers and are not capable of performing extraction as we have defined it. The robust statistics (RS) methods divide the geometric data into inliers and outliers, so they can be used for extraction purposes. We will show however, that there are limitations, some more apparent than real, in using these approaches for primitive extraction.

The most important measures used by the robust statistics community when discussing an RS approach are statistical efficiency and breakdown point. Statistical efficiency is the ability of an algorithm to correctly recover the characteristics of the original data. It is the traditional measure used to evaluate a fitting algorithm. Geometric data are usually modelled as consisting of the original uncorrupted data with noise added. When the distribution of the noise is Gaussian, then the least

squares estimator is known to be the most statistically efficient estimator possible [Walpole and Myers, 1989].

While useful, statistical efficiency is not a measure of robustness. According to the robust statistics literature this is measured by the breakdown point. The following explains the concept of breakdown in intuitive terms, while a more formal definition can be found on page 9 of [Rousseeuw and Leroy, 1987]. Take a set of points which are known to be described by a primitive (for example, a set of points which are all on a straight line) and perform a robust fit to these points. The resulting line is the baseline which is assumed to be correct. Now, one at a time, replace the good points by outliers (bad points which are far from the line) and perform the fit again using the same robust fitter. When the addition of a single outlier can make the computed fit arbitrarily distant from the correct fit, then breakdown has occurred. The breakdown point is the smallest percentage of outliers necessary to make this happen. The larger this percentage the more robust the algorithm.

According to the literature the maximum breakdown point of any robust fitter is 50% [Hampel *et al.*, 1986, Huber, 1981, Rousseeuw and Leroy, 1987]. We claim that while theoretically correct, the implications of this idea are not well understood. According to the definition of breakdown the robust fitter must be able to disregard any arbitrary configuration of outliers. This means that no characteristic of the inliers can be used to distinguish between them and the outliers, since the outliers could possibly align themselves to produce a better fit to the primitive than the inliers. This in turn implies that for robust fitting to be successful the inliers must form a majority [Hampel *et al.*, 1986, Huber, 1981, Rousseeuw and Leroy, 1987]. Thus the 50% limit is implied directly by the definition of breakdown.

While it is true that no estimator can have a breakdown point greater than 50%, it is also true that even though breakdown has occurred, this does not necessarily mean that the wrong estimate will be produced. For this to happen the outliers must actually align themselves to produce a better primitive than the inliers. Depending on the goodness of fit of the primitive to the inliers, and on the number of inliers and outliers, this may or may not happen. Of course, the larger the percentage of outliers

the more *likely* that this will occur. We show with a number of examples that it is indeed possible to tolerate more than 50% outliers, but not in the absolute sense used in the definition of breakdown.

In statistical applications robust fitters are normally used for regression problems, where there is only one primitive (or model) fit to the data. In this case the traditional definition of breakdown is natural, since if there is only one primitive and the fitting model is reasonable, then it is likely that at least 50% of the points will be inliers. However, for the primitive extraction problem there may be many geometric primitives in the data, and none may contain a majority of the total points. In such cases, a number of different primitives may be equally good choices.

We will show that the concept of breakdown is not as relevant to the computer vision field as to the statistics field. We believe that a more important issue in any robust process is deciding whether the results are significant. In other words, an extraction algorithm will always find a geometric primitive, but this does not mean that this primitive should be accepted as valid. This issue is dealt with in more detail in Section 2.5. We will show with a number of experiments that it is possible to have successful extraction with even 80% outliers. This kind of performance however, is possible only for accurate geometric data.

2.2.3 Some Robust Fitters

The least squares method is statistically efficient, but has a breakdown point of 0%; that is, a single outlier can cause the fit to be arbitrarily bad. The most common ways of achieving robustness in the RS literature use M-estimators [Hampel *et al.*, 1986, Huber, 1981]. The idea is to decrease the influence of outliers by weighting them less during the fitting process. Let $\overline{p}_1, \overline{p}_2, \ldots \overline{p}_N$ be the N input points, and let the geometric primitive be described implicitly by an equation of the form $f(\overline{p}; \overline{a}) = 0$. For this example we will assume that the error of fit (the residual) for geometric data point $\overline{p}_i$ is equal to r_i. To find the best fit using an M estimator it is necessary to find the parameter vector $\overline{a}$ that minimizes $\sum_{i=1}^{N} \rho(r_i)$. Here ρ is a positive definite function with a unique optimum at zero. If ρ is chosen to square the residuals then

the result is the least squares algorithm which minimizes $\sum_{i=1}^{N}(r_i)^2$ over $\bar{a}$. However, if ρ is chosen in such a way that points with a large error influence the fit less, then these points (which are likely to be outliers) will be ignored. This means that the robust fitting process using an M estimator is performing extraction, since the points which are outliers are marked as such.

Another common method for robust fitting is the Least Median of Squares (LMedS) algorithm [Rousseeuw and Leroy, 1987]. This has been developed for dealing with outliers during linear regression, and has been used by a number of vision researchers [Roth and Levine, 1990b, Roth and Levine, 1990a, Meer *et al.*, 1990, Kim *et al.*, 1989, Tirunmalai and Schunk, 1989]. The algorithm is best understood by again considering the case of fitting a line to a set of N points. Two points are required to uniquely define a line. The algorithm randomly selects K sets of two points (how K is chosen can be found in section 3.1). For each of the lines defined by these two points the residuals (errors) of all the N points relative to this line are computed and squared. Then the median of these squared residuals is found (the median is the middle element of the sorted squared residuals) and associated with this particular set. The set which has the least median squared (LMedS) error is the one which will be used to decide which of the N points are inliers and which are outliers. The inlier/outlier selection is made by examining the squared residuals for the line associated with the set and discarding as outliers those whose residual is too large. The threshold used to discard outliers is labeled T, and is commonly set to $2.5 \times V$, where V is the noise variance of the data. The reasoning behind this is that any point which is more than two and one half times the distance of the noise variance from the geometric primitive is likely to be an outlier. These two robust fitters are the most common ones used in the computer vision field.

Another recent example of a robust fitter that has been applied to a number of computer vision problems is the MF estimator, which is an abbreviation of "model fitting" [Zhuang *et al.*, 1992]. Here the log likelihood function of the unknown distribution is modelled. This partial modelling takes place in terms of the Bayesian statistical decision rule. This rule, along with a number of important heuristics are

added to produce a procedure for actually computing the value of this estimator. This work is similar to ours in that more robustness than the standard robust statistics fitting procedures is claimed for the MF estimator. However, this issue has not been addressed thoroughly. There is also little discussion of why the algorithm for computing this estimator is successful. Thus, while somewhat similar to our work, this is not a consensus-based approach.

As we said in the introduction of this chapter, the robust statistics community naturally concentrates on the statistical properties of various robust estimators. This means that given a set of statistical assumptions, the maximum likelihood, or Bayesian approach, is used to find an expression whose optimum value must be computed to perform the estimation. However, the issue of how to find an efficient computational method to actually perform this optimization is often not discussed in any detail. This is what we concentrate on, and we will show by the optimization model that we present in the next section that the kind of optimization problems associated with primitive extraction have a certain commonality.

2.3 Optimization Model for Extraction

In this section we will describe our optimization model for primitive extraction. It has long been known that fitting a geometric primitive to geometric data can be framed as an optimization problem. What we will show is that not just fitting, but extraction, which is a generalization of fitting, can also be viewed in this optimization framework.

As we have noted previously the inputs to an extraction algorithm are N geometric data points labelled $\overline{p}_1, \ldots, \overline{p}_N$, along with the definition of a geometric primitive in implicit form. The output consists of the parameter vector $\overline{a}$ of the best primitive, along with the subset of the geometric data that belongs to this primitive (the inliers). According to our optimization model the first step in any extraction algorithm is the computation of the residual, which for point i of the geometric data is labelled r_i. For a function f, which is a graph function, this residual is equal to $f(\overline{p};\overline{a}) - \overline{p}$. However,

for other types of functions it is desirable to use the closest distance of the given point to the primitive as the residual value. For planes and other simple primitives this distance can be computed exactly in a closed form. However, the closed form solution for the closest distance for more complex geometric primitives is so unwieldy as to be impractical [Nalwa and Pauchon, 1987, Pratt, 1987]. In fact, the problem of finding the closest distance to a curve or surface is difficult for both the parametric and implicit form [Besl and McKay, 1992].

For this reason the first order approximation to the closest distance is often used in its place [Bookstein, 1979, Sampson, 1982, Taubin, 1991]. For the primitive defined by implicitly by a function f the first order approximation of the closest distance of a point $\overline{p}$ to the curve or surface is the absolute value of the function over the magnitude of the gradient vector ($\approx |f(\overline{p};\overline{a})|/|\nabla f(\overline{p};\overline{a})|$). This expression is simple to calculate and is frame-invariant, since the gradient operator produces the same vector in any coordinate system. For primitives other than lines or planes this value is only an approximation. However, as long as the point $\overline{p}$ is not too far from the curve or surface the error is not significant. This function is adequate for our algorithm because we require accurate estimates of the residuals only when a given point $\overline{p}$ is close to the curve or surface. This is because for the most common class of cost functions, the fixed-band cost functions, the residual must be computed accurately only for points within the band size, which is normally a small distance. Computing these residuals more accurately is difficult and computationally intensive [Besl and McKay, 1992]. We use this approximation to compute the residuals for all the examples in the thesis and have found it to be sufficient for this application.

Given that residuals $r_1,\ldots,r_N$ have been calculated, then an extraction algorithm must find the parameter vector $\overline{a}$ which optimizes the value of a cost function $h(r_1,\ldots r_N)$. The cost function returns a scalar which measures the "goodness" of the primitive with the given parameter vector. The only restriction we make on the cost function is that $h(r_1,\ldots r_N) \geq 0$ for all $\overline{a}$, and that $h(r_1,\ldots r_N)$ attains its optimum value when the residuals $r_1,\ldots,r_N$ are all zero. Since each of the residuals is a function of the parameter vector $\overline{a}$ this implies that h is an indirect function of $\overline{a}$.

The goal of primitive extraction is to find the parameter vector $\bar{a}$, which optimizes the cost function, that is to $\min_{\bar{a}} h(r_1, \ldots r_N)$ or $\max_{\bar{a}} h(r_1, \ldots r_N)$ depending on the cost function.

With this model it is possible to describe many different extraction and fitting methods by using different cost functions. The following is a list of a number of extraction and fitting methods, along with their cost functions:

- Least Squares: $h(r_1, \ldots, r_N) = \sum_{i=1}^{i=N} {r_i}^2$.
- M-Estimation: $h(r_1, \ldots, r_N) = \sum_{i=1}^{i=N} \rho({r_i}^2)$ where ρ is a symmetric positive-definite function with a unique optimum at zero.
- Least Median of Squares: $h(r_1, \ldots, r_N) = \text{median}(r_1^2, \ldots, r_N^2)$.
- Template Matching (HT): $h(r_1, \ldots, r_N) = \sum_{i=1}^{i=N} s({r_i}^2)$, where s is step function; $s = 1$ if r_i is greater than or equal to the template width, and $s = 0$ otherwise.

We will now describe each of these cost functions in more detail. The least squares approach is the usual non-robust fitting algorithm. It is well known that the addition of a single extraneous point (an outlier) to the input can cause the resulting primitive to be an arbitrarily bad fit. The M-estimation approach [Hampel *et al.*, 1986] and the Least Median of Squares (LMedS) algorithm [Rousseeuw and Leroy, 1987] are two methods from the RS field which overcome this lack of robustness. When using M-estimators, ρ is chosen in such a way that points with large errors influence the fit less, so these points (which are likely to be outliers) are ignored. The LMedS algorithm is another approach from the RS field, but it is not based on M-estimators. Instead, the median of the square of the residual values is chosen as the value of the cost function. If the outliers constitute less than 50% of the geometric data, then this median operation will discard them. The HT is known to be equivalent to template matching, where the templates are defined by each of the cells in parameter space. If the cell size is small enough the templates are of fixed size and shape, and the HT can be described by the optimization model by using the appropriate cost function. In this case the cost function reflects the number of geometric data points inside a template of a given width centered on the geometric primitive. It is clear that this

optimization model can describe various fitting and extraction algorithms by simply using different cost functions. We say a cost function is robust if it can be used for extraction. Such a cost function must be able to mark some of the points as inliers, and the rest as outliers. A cost function which is not robust can only be used for fitting, since it assumes all the points are inliers. The first cost function we have listed, the least squares cost function, is not robust, while the other three cost functions are robust.

For the robust cost functions there are potentially many local optima, which is not the case for the least squares cost functions. Each local optima corresponds to a different partition of the N geometric data points into inliers and outliers. The inliers are points that are taken into account by the cost function, and the outliers are ignored. We say that a geometric data point is marked as an outlier if its residual can be increased to any arbitrary value, without changing the cost function value. By contrast, an inlier is a geometric data point which would change the cost function value if the residual of this point were increased, even if this change is infinitesimal. The best such partition into inliers and outliers is the one at which the cost function has the global minimum. Since there are many such possible partitions, this means that the maximum possible number of such local optima is very large. This maximum number of local optima is equal to the number of ways of choosing a subset of size M or less from N points, where M is the maximum number of possible outliers. M can take any value from 1 to $N - R$, where R is the size of the minimal subset. Thus the maximum number of local optima is equal to the number of possible subsets, which is $2^{(N-R)}$. This number of local optima is only possible if the number of outliers is equal to the maximum value $(N - R)$. However, even for a small number of outliers, the number of local optima for a robust cost function may be very large.

Next we will consider a simple example which shows the multi-modal nature of a robust cost function. In Figure 2.3 are shown two lines that intersect at the origin. The lines are parameterized in terms of ρ and θ, where ρ is the distance of the line from the origin, and θ is the angle relative to the x-axis. In this case the equation of the line is $x\cos(\theta) + y\sin(\theta) - \rho = 0$. For both the lines in the figure ρ is zero since

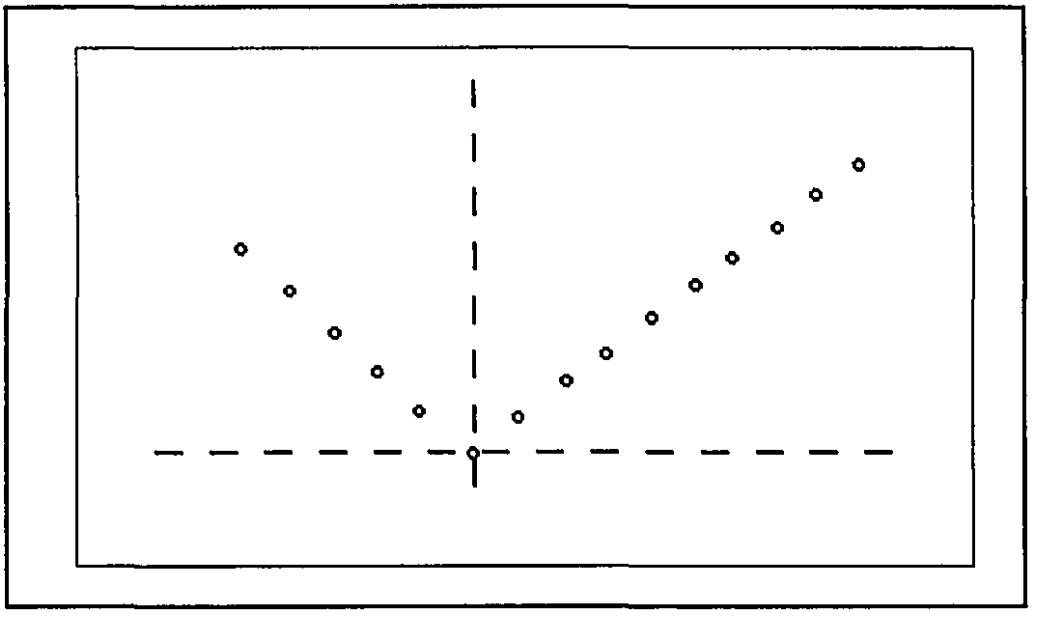

Figure 2.3: Two orthogonal lines in the plane.

the lines pass through the origin. One of the lines has a θ of $\frac{\pi}{4}$ radians, while the other line has a θ of $\frac{3\pi}{4}$ radians. Figure 2.4 shows a three-dimensional plot of the cost function value for a Hampel influence function versus ρ and θ, the line parameters. From this figure we see that the cost function value has two local optima. Each one of these optima corresponds to one of the $2D$ lines. For both optima the ρ value is zero, and they have the appropriate θ values for each line. The global optimum is when θ equals $\frac{\pi}{4}$, which accords with the geometric data, since this line has the most points.

We said that a point is an outlier if its residual can be increased arbitrarily without changing the value of the cost function. This implies that for a given value of the parameter vector $\overline{a}$ the following procedure can be used to label each geometric data point as an inlier or outlier. For the given $\overline{a}$, compute r_1 through r_N, and then compute $h(r_1, \ldots, r_N)$. Now, for geometric data point i, set the residual r_i to ∞, and recalculate the value of the cost function. If this new value of h is the same as the value with the original r_i, then this point is an outlier; otherwise it is an inlier. The reasoning is that if the cost function value did not change when the residual increased to ∞, then this point has been ignored, and is therefore an outlier. Repeating this

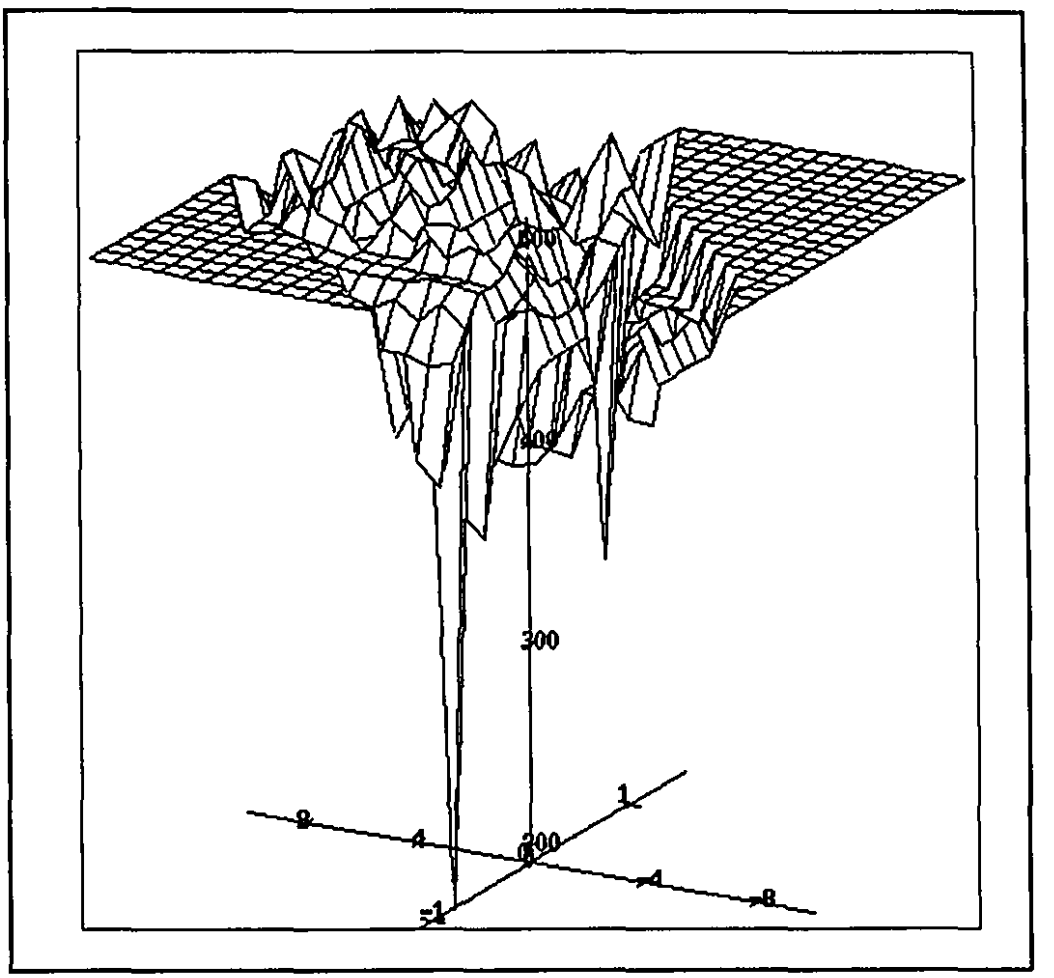

Figure 2.4: Hampel cost function value for extraction of two lines: Axes are ρ and θ value of the line.

process for each input point categorizes the points as either inliers or outliers.

This optimization model shows the essential unity of various methods for fitting and extraction. For a fitting algorithm the assumption is made that all points are inliers, so a non-robust cost function such as least squares can be used. On the other hand, for extraction this assumption does not hold, and a robust cost function must be used. There are many different robust cost functions, however, they all have in common the fact that they usually contain many local optima. The goal of an extraction algorithm is to find the global optimum of a cost function which may have very many local optima. Because the number of local optima is often very large, extraction is an inherently difficult problem. This is especially true in comparison to fitting, where the number of local optima is very small (often one). Since there are N residuals to calculate, where N is the number of geometric data points, a single evaluation of a cost function requires at least $O(N)$ time. Since N may be large an efficient extraction algorithm must be able to find the global minimum, or a value close to it, with as few evaluations of the cost function as possible.

2.4 Random Sampling for Primitive Extraction

One way to find the global optimum is to evaluate the cost function at the locations in the parameter space defined by a repeatable grid. This is exactly what the HT does by computing the cost function at the values of the parameter vector $\overline{a}$ associated with each cell in the parameter space. How confident we are that this procedure finds the true global optimum depends on the cell size. The smaller the cells the more evaluations of the cost function, and the more the confidence. As the cell size approaches zero the number of cost function evaluations approaches infinity. This brute-force approach can clearly be improved upon. It is also very inefficient for primitives with high dimensional parameter spaces.

In the computer vision literature there are two widely used methods for finding the global optimum of functions with many local optima: continuation methods [Blake and Zisserman, 1987, Leclerc, 1989] and stochastic methods

[Corana *et al.*, 1987]. While continuation methods are effective, they assume a continuous cost function, which is too restrictive for our optimization model. They also do not actually guarantee convergence to the global minimum, even for perfectly accurate data. For these reasons continuation methods will not be discussed further. The other approaches to finding the global optimum are stochastically based, which means the cost function is evaluated at random locations in the parameter space. The best known of these stochastic methods is simulated annealing, which is derived from an analogy to the physical process of annealing. It has been shown that if the parameters of the simulated annealing process are carefully chosen, then convergence to the global optimum can be theoretically guaranteed. The main drawback is that the number of evaluations of the cost function required to find the global optimum is very high. Since evaluating the cost function is computationally expensive this is a significant disadvantage. For this reason we do not use simulated annealing to solve the optimization problem that we have posed.

Given the fact that our cost function need not be continuous, it is clear that some kind of stochastic process must be used to find the global minimum. Our approach is to evaluate the cost function only at the values of the parameter vector defined by randomly chosen minimal subsets of the geometric data. A minimal subset contains the smallest number of points necessary to unambiguously define the geometric primitive. If the parameter vector $\overline{a}$ of the primitive's implicit function f has dimension $R+1$, then f has R degrees of freedom. The reason that the dimension of $\overline{a}$ is one greater than the degrees of freedom is that for a given point $\overline{p}$, if $f(\overline{p};\overline{a})$ is zero, then $c \times f(\overline{p};\overline{a})$ is also zero for any constant c. R points are necessary and sufficient to unambiguously define a finite number of geometric primitives through these points. Any such R point subset of the geometric data is called a minimal subset. For example three points are necessary and sufficient to define a circle. With two points the circle is underconstrained, while with four points the circle is overconstrained. In this case there is exactly one circle defined by the minimal subset. While there is often only a single primitive through a minimal subset, this is not always the case. For example, if the circle has a known radius then the minimal subset is of size two,

since there are potentially two circles with the given radius. However, the number of possible geometric primitives defined by a minimal subset is finite, and usually very small.

There is a significant advantage to evaluating the cost function only at the values of the parameter vector defined by each minimal subset. This is the fact that there are only a finite number of such subsets, and this number is equal to $\binom{N}{R}$. For perfectly accurate data, evaluating the cost function at all the minimal subsets is guaranteed to find the global minimum. Since $\binom{N}{R}$ is bounded by N^R the number of cost function evaluations is therefore a polynomial function of the number of geometric data points N. This is the main theoretical justification for the minimal subset approach. Other methods of finding the global optimum cannot give any such bounds on the number of cost function evaluations, even for perfectly accurate geometric data. We will show that it is rarely necessary to evaluate the cost function for all the minimal subsets in order for successful primitive extraction. A much smaller number of randomly chosen minimal subsets is usually sufficient.

The pseudo-code for the basic random sampling extraction algorithm follows. The input is a set of N points, the definition of the cost function h, and the primitive to be extracted in implicit form.

For K randomly chosen sets of R points
Do

1. *Find the parameter vector $\overline{a}$ of the primitive through all of the R points.*
2. *Compute the residuals $r_1, \ldots, r_N$ of the entire set of N points with respect to the geometric primitive defined by $f(\overline{p}; \overline{a})$.*
3. *Rank the goodness of this primitive by evaluating the cost function $h(r_1, \ldots, r_N)$.*
4. *Save the primitive with the smallest cost along with the associated parameter vector $\overline{a}$.*

Enddo

This basic methodology is a generalization of the resampling algorithms from the RS field [Rousseeuw and Leroy, 1987]. The output consists of the parameter vector of the best minimal subset, along with the cost function value. This information, along with the definition of the cost function h, enables each of the N geometric data points to be marked as an inlier or an outlier by the method described in Section 2.3.

The time complexity of the algorithm is at best $O(KV)$, where K is the number of randomly chosen minimal subsets, and V is the time taken to evaluate the cost function. We assume that there N geometric data points. We will show that depending on the particular cost function the evaluation time is usually $O(N)$, or $O(N \log N)$. The space complexity is usually $O(N)$, but for certain cost function it may be larger. In the next sections we will consider each of the important parts of the algorithm in more detail. In this chapter we will ignore the issue of how to convert between a minimal subset and a parameter vector which is essential for Step 1 of the algorithm. For now, we will simply assume that this ability exists. This procedure, which is instrumental to our approach, will be discussed in detail in Chapter Three.

2.4.1 Choosing K, the Number of Random Samples

The first question we will consider is how to select K, the number of randomly chosen minimal subsets. The maximum value of K is $\binom{N}{R}$, but as we have stated this number is too large for any reasonably sized N. We will show that in most cases a value of K much less than the maximum number is acceptable.

The value of K depends on the minimum number of geometric data points that are expected to be on a single valid geometric primitive. It is usually possible to set this number (which we label as Y) directly from the significance threshold of the cost function, and this procedure is discussed in detail at the end of Section 2.5. Let ϵ be the probability of a single randomly drawn geometric data point being on the desired primitive. The value of ϵ is no less than Y/N, where Y is the minimum number of points that must be on a single geometric primitive for it to be accepted as valid, and N is the number of geometric data points. The probability of all of the R randomly drawn elements of a single minimal subset being on the primitive, that is being inliers, is ϵ^R. The probability of at least one of the set of K minimal subsets being all inliers is labelled s, and s as a function of ϵ, R, and K is:

$$s = 1 - (1 - \epsilon^R)^K \tag{2.1}$$

This formula is a simple application of combinatorial analysis and the same result has been presented elsewhere [Rousseeuw and Leroy, 1987, Fischler and Bolles, 1981]. The value of K as a function ϵ and R is:

$$K = \frac{\ln(1-s)}{\ln(1-\epsilon^R)} \tag{2.2}$$

If we wish to have a high confidence of successful extraction then s is set to a large value (usually .95), and K is chosen accordingly. However, this is a worst-case value for K. The expected value for K, which can be found by setting s equal to .5, will be less.

For most values of ϵ the value of K required for successful extraction is much less than $\binom{N}{R}$. This is demonstrated in Table 2.1 which lists the required K for the case of $\epsilon = .5$ (50% inliers) to reach 95% and 50% confidence of success ($s = .95$ and $s = .5$). That so few minimal subsets are necessary is surprising. The explanation is that K as defined in the previous equation exhibits a definite threshold effect, being close to $\binom{N}{R}$ only for small values of ϵ. This is demonstrated in Table 2.2 when ϵ is computed for $s = .5$ and $s = .95$ with varying K and R. This table shows what fraction of inliers we are sure to find (95% confident), and what fraction we are likely to find (50% confident) with K randomly sampled minimal subsets. For example, assume we are searching for a circle, which makes R equal to 3. Then if K is set to 640 we see from Table 2.2 that we are 95% certain of finding a circle with at least 16% of all the points (ϵ of .16) and 50% certain of finding a circle with at least 10% of all the points (ϵ of .10). This table demonstrates that for many values of ϵ the required value of K for successful extraction is not excessive.

If it is not possible to estimate ϵ, then there is no principled way to set K. All that can be done is to take as many random samples as are practical. However, the previous formulae can still be used to compute what size of primitive should have been successfully extracted with K samples. Thus again in relation to the previous example, if we are searching for a circle and have taken 640 randomly sampled minimal subsets, then we are 95% certain that we will find any circle that has at least 16% of

Size of minimal subset R	*Number of Trials K*	
	$s = .95$	$s = .50$
1	5	2
2	11	3
3	23	6
4	47	12
5	95	23
6	191	45
7	382	89
8	766	178
9	1533	356
10	3067	710

Table 2.1: The number of trials K for 95% and 50% confidence with 50% inliers for a given size of minimal subset R.

No. of Subsets K	ϵ *for* $s = .95$ *and* $s = .50$ *confidence*			
	R = 2	R = 3	R =4	R = 5
10	.50 .25	.63 .40	.71 .50	.76 .58
20	.37 .18	.51 .32	.61 .42	.67 .50
40	.26 .13	.41 .25	.59 .44	.59 .44
80	.19 .09	.33 .20	.43 .30	.51 .38
160	.13 .06	.26 .16	.36 .25	.45 .33
320	.09 .04	.21 .12	.31 .21	.39 .29
640	.06 .03	.16 .10	.26 .18	.34 .25
1280	.04 .02	.13 .08	.21 .15	.29 .22
2560	.03 .016	.10 .06	.18 .12	.25 .19
5120	.02 .010	.08 .05	.15 .10	.22 .16
10240	.01 .008	.06 .04	.13 .09	.19 .14

Table 2.2: The fraction of inliers on a single primitive (ϵ) found with 95% and 50% confidence with K random subsets for R from 2 to 5.

the total number of points.

2.4.2 Selecting and Evaluating the Cost Function

In step 3 of the basic algorithm we must rank the primitives defined by each of the minimal subsets. In order to do this a particular cost function must be selected. The input to this cost function are the residuals of all the geometric data, which depend on the parameter vector $\overline{a}$. The cost function returns a scalar which measures the "goodness" of the primitive with a particular parameter vector. The lower the cost

function value, the better the primitive. In Section 2.3 we have listed a number of cost functions, but this list does not exhaust the possibilities. Even though there are many possible cost functions in our experience most of them can be classified into two categories, fixed-band or variable-band. Each of these categories makes different assumptions about the geometric data.

We say a cost function is in the fixed-band class if any point whose residual value is greater than a fixed threshold is marked as an outlier. This threshold defines what we call a fixed-band around the geometric primitive. Outside this band are the outliers, and inside this band are the inliers. In many applications the band size can be sensibly set from the variance estimate of the noise present in the geometric data. This in turn can usually be obtained from an analysis of the sensor and the data creation process [Amid *et al.*, 1988]. In our experiments we have found a fixed-band cost function produces good results. The influence functions in the RS field are an example of a fixed-band cost function, as is the cost function used by the HT.

If such a variance estimate is not available then a variable-band cost function is a better choice than a fixed-band cost function. An example of such a function is the Least Median Squared algorithm (LMedS) [Rousseeuw and Leroy, 1987]. Here, the assumption is made that at least half of the points belong to a single primitive. Assume the squared residuals of the N points are sorted from smallest to largest. Then let r_l be entry number l in this sorted list of squared residuals. For the LMedS algorithm l is set to $N/2$, which means that the cost function $h = median(r_1^2, \ldots, r_N^2)$. This approach can be easily adjusted to situations where a single primitive is known to contain not half the geometric data, but any specified fraction. This is done by setting l to the minimum number of expected inliers in a single primitive, instead of using the value of $N/2$. Such a cost function is equivalent to a variable-band whose size is adjusted to contain exactly the required percentage of inliers. In the same way as for a fixed-band cost function, inside this variable band are the inliers, and outside it are the outliers.

For the fixed-band cost function the user sets the band size, and for the variable-band cost function the user sets the required percentage of inliers. In both cases the

user must decide whether the extraction results are significant by looking at the cost function value of the returned best primitive. The fixed-band cost function value indicates the number of inliers on the best primitive, while the variable-band cost function indicates the band size of the best primitive. Thus the fixed-and variable-band cost functions are duals of one another.

2.4.3 Speeding up the Evaluation Process

From the pseudo-code of the random sampling algorithm it is clear that the most expensive step in our algorithm is the evaluation of the cost function. If there are N points, then this operation seems to requires at least $O(N)$ time. For fixed-band scoring the cost function evaluation can be clearly done in $O(N)$ time for both the average and worst case. The variable-band cost function described in the previous section requires the computation of the element in position l of the squared residuals, sorted from the lowest to the highest value. This is known as an order statistic and algorithms exist to find these in of $O(N)$ time on the average [Aho *et al.*, 1975]. It is clear that since N, the number of geometric data points may be quite large, evaluating the cost function is a potential computational bottleneck.

We will show that there are ways of lowering the time complexity for cost function evaluation. We will described the following methods of achieving this goal:

- Prior Constraints on the Primitive
- Using Primitive Generation Routines
- Local Gradients
- Subsampling
- Hierarchical Data Structures

Note that none of these methods use parallel hardware. Using such hardware to speed up the cost function evaluation is an issue that we will explore in detail in Chapter Six.

Ideally we would like to be able to discard a primitive through a minimal subset without evaluating the cost function. There are two ways this can be done. The first way is to use a priori constraints on the primitive, and the second is to use local gradients. Both those methods are important because avoiding the cost function evaluation stage can provide a dramatic increase in performance. This is because generating the equation of a potential primitive from a minimal subset is usually much faster than actually evaluating the cost function for the primitive, especially for large N.

There are often prior constraints on the size or orientation of a primitive that is to be extracted. For example, if the primitive is a circle it may be that the circle center is known at least approximately, or that the approximate radius of the circle is known. Then any circle produced by a minimal subset which does not meet this constraint can be discarded without evaluating the cost function. A possible constraint for a line is that its angle relative to one of the axis be in a given range, and lines which do not meet this angle constraint could be discarded. By using such prior constraints to discard a minimal subset we are able to speed up the extraction process considerably.

There is a method which has considerable potential to speedup the fixed-band cost function evaluation for 2D curves when the sensor data consists of 2D geometric data, such as the edge points obtained from an intensity image. The extraction of 2D curves such as lines, circles and ellipses, from such data is a common problem. This approach can be applied to these and other types of 2D curves. In many curve extraction applications the cost function that is used most often simply counts the number of geometric data points within a small band of fixed-size around a given curve. This type of cost function is equivalent to traditional template matching, and is a fixed-band cost function. The obvious way to do this matching is to compute the distance of each of the N data points from the curve, and then to count all the points that are within the fixed-band. Using this method the time taken to evaluate the cost function is $O(N)$.

Our idea is to match these curves against the 2D geometric data by using an incremental curve generation algorithm. Such algorithms are widely used for drawing

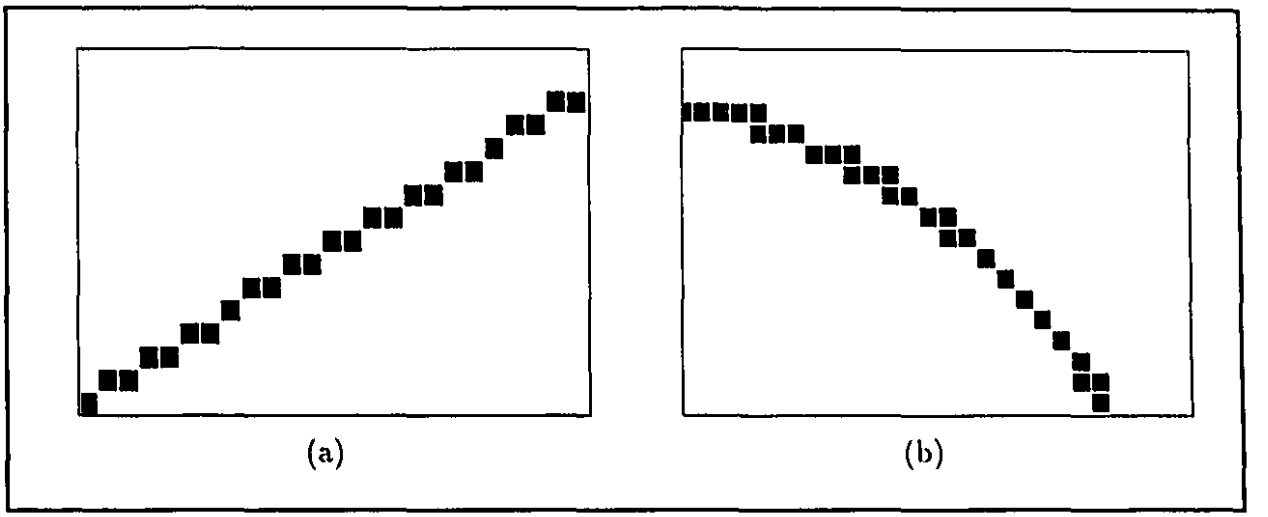

Figure 2.5: Incremental drawing routines (a) Pixels for a line (c) Pixels for an arc of a circle.

2D curves on the bit-map screens of graphic displays [Hegron, 1988, Bresenham, 1977, Bresenham, 1965, VanAken, 1984]. Two examples of their output are shown in Figure 2.5. First we take the list of 2D geometric data points and map them onto a binary 2D array. This is done by setting the associated array element of the geometric data point to one, and setting all the other array elements to zero. Using this array we can compute the cost function value for each hypothesized curve by executing an incremental curve generation routine.

More precisely, the input to this routine is the equation of the curve produced by the random sampling process, and the binary array containing the 2D geometric data points. The curve generation routine then produces the array indices of each point on the curve. This is done by moving along the curve to be matched from point to point in a very efficient manner. Normally these routines are used to draw the curve by setting each pixel in the associated binary array to one. Instead of drawing the curve, we simply count the number of array points that are also curve points. This count represents the number of 2D geometric data points that are within the fixed-band template around the curve. If the template is more than one pixel in width we simply generate the indices of the pixels for a thicker curve [Wallis, 1990]. However, as the template width increases more and more points will be on the curve. Therefore this approach will be efficient only if the fixed-band size is small (four pixels or less).

However, for many extraction applications from 2D geometric data this is indeed the case. The reason is that these 2D geometric data points are usually produced by the process of edge detection in intensity images. Such edges are normally at most a few pixels in width.

With this approach the matching time is $O(Z)$, where Z is the number of pixels on the curve, instead of $O(N)$, where N is the number of the geometric data points. The value of Z depends on the curve and the band size. However, for most curves and band sizes, Z is fairly small. For example, assume the edge points are produced on a standard 512 by 512 intensity image. Then for line extraction with a fixed-band size of one pixel Z, the maximum number of points on the line is 750. The main advantage of this method of template matching is that Z is independent of N, the number of geometric data points. This means that as N increases the execution time of this method of template matching does not increase. In other words, as long as the percentage of points on the curve stays the same, the time taken to extract the curve does not change as the number of geometric data points N increases.

There are available efficient incremental drawing routines for lines, circles, parabolas, and ellipses [Hegron, 1988, Bresenham, 1977, Bresenham, 1965, VanAken, 1984]. In fact, some of them have been implemented in VLSI hardware [Asal *et al.*, 1986]. In principle, the same incremental drawing approach can be used for any curve defined by an implicitly [Chandler, 1988]. On serial computers the typical rate at which such curves can be matched is in the order of several hundred per second, while with VLSI implementations this increases to several thousand. Using fast VLSI drawing hardware for curve matching in this fashion has the potential to achieve real-time performance for matching. This means that the primitive extraction algorithm could also be done in real-time for these types of curves, since the matching phase is the most computationally intensive part of the extraction algorithm.

Another way to discard a primitive without evaluating the cost function is to use local gradient information. It is often possible to obtain a local gradient estimate for each geometric data point. For example, if the geometric data consist of edge points in an intensity image, then these points were produced by thresholding the output of

a local gradient operator. In this case the gradient vector is a direct by-product of this operator. For three-dimensional data it may be the case that the local surface normal (the three-dimensional gradient vector) is available in a similar way.

When available, local gradient estimates computed from the geometric data can be compared to the predicted gradient from the implicit form of the geometric primitive. Then the cost function evaluation need be done only if these two estimates are approximately the same. The computed gradient is obtained from the geometric data; the predicted gradient is obtained from the implicit function f. The gradient vector consists of the partials derivatives of f with respect to x, y (and z for three-dimensional data). This is the predicted gradient vector, and for a given geometric data points it should be approximately the same as the computed gradient vector. For perfect data the match should be exact; in reality the degree of acceptable difference between the two is a threshold set from the resolution of the geometric data. The utility of this approach depends on the accuracy of the geometric data; the more accurate the better the gradient estimates which means that more minimal subsets can be discarded in this fashion.

Another way to speed up cost function evaluation is by subsampling the geometric data. This means using only a subset of the N geometric data points (say Q points out of N) to evaluate the primitive. These Q points might be chosen randomly from the N points, or selected in any fashion that is appropriate. If Q is considerably less than N then the decrease in execution time is substantial. This idea has been applied previously to the HT [Kiryati *et al.*, 1991]. However, the optimal choice of Q, and the resulting computational savings are problem dependent. It has been experimentally found that for complex images, Q may need to be as large as $N/2$, while for simple images Q can be as small as $N/10$. For this reason we do not advocate this method of speeding up the evaluation of the cost function.

Another method that can be used to speed up the evaluation of the cost function is the use of a hierarchic data structure. The idea is to use such a structure to avoid processing all N geometric data points during the evaluation of the cost function. In this way, the time taken to evaluate the cost function decreases. Such a data structure

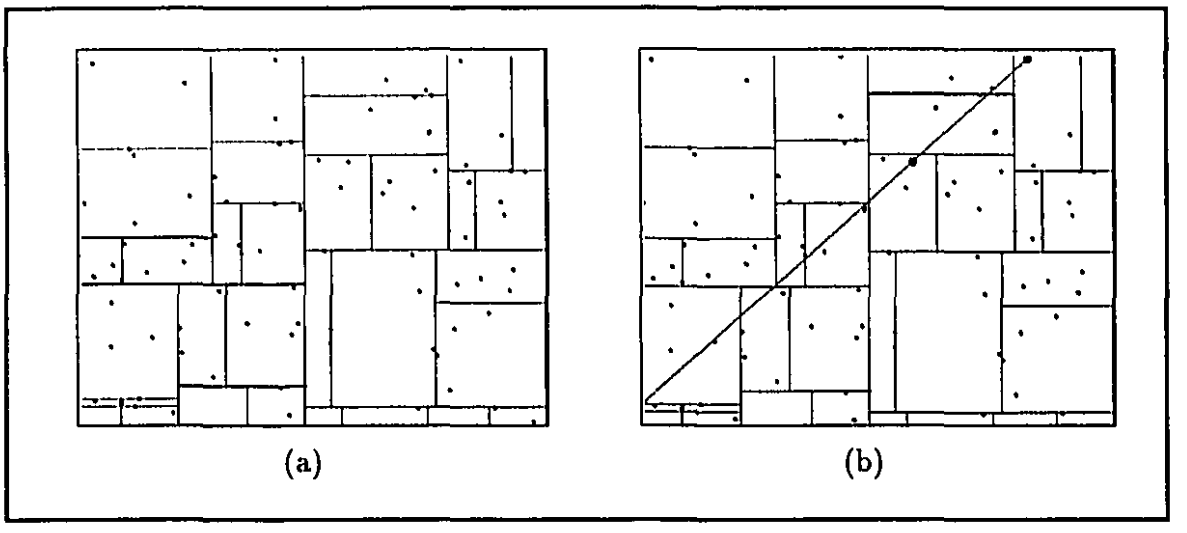

Figure 2.6: A kD tree for a set of points (a) the recursive tree structure (b) a line superimposed on the tree structure

describes the geometric data in a hierarchical fashion. The top node in the hierarchy contains all the data, which are subdivided in a recursive fashion until finally the actual points themselves are obtained [Aho *et al.*, 1975]. The data structure we have chosen to demonstrate this concept is a kD tree, but the idea will work with any hierarchical data structure. A kD tree is a recursive binary tree used for storing and operating on records containing k dimensional keys. In our case, the keys are simply the N geometric data points. The creation of such a data structure from the geometric data points is described in the literature [Samet, 1984]. In figure 2.6 part (a), we show part of a kD tree for a set of two-dimensional points. The root of the tree is the entire set of points, and each node starting from the root has two children. Each of these children are subtrees that contain exactly half of their parents points. This recursive subdivision process is normally repeated until a node contains a single point. However, in this example we show the nodes only to a fixed depth in the tree.

In figure 2.6, part (b), a line obtained by the random sampling process is superimposed on the kD tree structure. We assume that a fixed-band cost function is used to evaluate the line. This means that the number of points that are within a fixed distance of the given geometric primitive must be counted. Normally this requires the computation of the distance of each point to the primitive to decide if it is within

the fixed-band. However, we will show that with the kD tree structure this is not the case.

Consider the corners of the bounding box of any given node of the tree as shown in figure 2.6 and their relationship to the fixed-band surrounding the primitive. For now, we will assume that the band size is a single pixel. If the corners of the box are on the same side of the primitive, then none of the points in this node or its descendents can be in the fixed-band. This means that evaluating the cost function requires not $O(N)$ time, but $O(R+log(N))$ time, where R is the expected number of points in the fixed-band. This complexity figure is derived from an analysis of the cost of searching a kD tree [Samet, 1984]. For a small band size the expected number of inliers R is considerably less than N, so at least in the asymptotic case (as N becomes large), this is a significant reduction in the time complexity. For the variable-band case, the time complexity is similarly reduced to become $O(log(S) \times (R + log(N))$, where S is an upper bound on the variable-band size. The idea here is to do a binary search at different band sizes until l points are in the band, where l the required number of points.

Of course, whether such a data structure is warranted depends on the number of geometric data points N. The complexity figures only guarantee that as N increases there will always be a point where using kD tree becomes more efficient. However, this crossover point can only be determined experimentally by comparing the running times of actual implementations with and without kD trees. The use of a kD tree shows that it is not strictly necessary to interrogate all N points to evaluate the cost function for certain types of geometric primitives.

2.5 Robustness of Extraction Algorithms

An issue which was discussed previously is that of robustness. We have stated in Section 2.2.2 that the robust statistics concept of breakdown is inadequate for computer vision purposes. There, we made the claim that in practice a robustness of more than 50% is indeed possible. In this section we will discuss the issue of robustness in more

detail. The robust statistics community has consistently stated that the maximum possible robustness is 50% outliers, and has used the concept of breakdown to justify this claim [Hampel *et al.*, 1986, Huber, 1981, Rousseeuw, 1984]. More recently, an analysis of the HT has concluded that it can deal with more than 50% outliers, but not with the case where the inliers are but a very small fraction of the input data [Grimson and Huttenlocher, 1990]. We claim that the HT and the influence function methods can both be described by our optimization model with the appropriate fixed-band cost function. Since the cost functions are both of the same type, the maximum possible robustness of both approaches should be equal, which does not agree with the literature [Forstner, 1989].

For fixed and variable-band cost functions we will show that experimentally a robustness of greater than 50% is indeed possible, but not in the sense defined by the idea of breakdown. The question then is what is the maximum possible robustness that can be achieved by a robust extraction method? To answer this question properly we must look at the question of robustness from a different viewpoint. A robust extraction procedure will always return a result for any given problem. We believe that for primitive extraction the important issue is not robustness as defined by the concept of breakdown, but how to decide on the significance of the result. Clearly as the percentage of outliers increases, at some point the result will not be significant. Thus for fixed and variable-band cost functions, we define the maximal robustness as the largest percentage of outliers for which the result is insignificant.

One way to decide on significance is by using a simple statistical model. Assume that our extraction procedure finds a given geometric primitive with a particular cost function value. There are a number of inliers; that is geometric data points that belong to this primitive. Let us assume that the geometric data points are uniformly distributed. Then the probability of this many inliers being randomly aligned in such a way that their cost function value is greater than or equal to that of the extracted primitive can be computed. To compute this probability it is first necessary to compute the probability of a single uniformly distributed geometric data point being an inlier. This is equal to the band size of the cost function (fixed

or variable-band) divided by the total volume or area covered by the sensor. This computation requires a knowledge of the sensor and its operating parameters along with the definition of the geometric primitive, and of course the band size.

The following example illustrates the approach. For a given cost function h, we let B equal the inlier band size divided by the total area (or volume) covered by the sensor. If the geometric data is distributed uniformly, then the probability of a single point randomly falling in this inlier band is B. Given N geometric data points, the probability of exactly t of them being inliers is given by the following binomial distribution:

$$p_t = \binom{N}{t} B^t \Big(1 - B\Big)^{N-t} \tag{2.3}$$

This means that the probability of l or more inliers occurring at random ($p_{\geq l}$) is equal to:

$$p_{\geq l} = 1 - \sum_{t=0}^{l-1} p_t \tag{2.4}$$

Using this last formula (which was also derived in [Grimson and Huttenlocher, 1990] and in [Stewart, 1991]) it is possible to compute the significance of any particular extraction result.

As an example, consider the situation where the goal is to extract a line from edge points. Assume the input consists of edge points from a 2D image in which the sensor area is 512 by 512 picture elements (pixels). Table 2.3 shows $p_{\geq l}$, the probability of an accidental alignment of l or more inliers when B is computed for a fixed-band cost function of width one to four pixels. In this example the number of geometric data points N, is 1000. For line extraction with a fixed-band cost function the value of B is easily computed. For example, if the fixed-band has a width of two pixels, the total area of this inlier band is 1024 pixels, assuming a maximum line length of 512 pixels. This makes B the probability of a single point falling randomly in this inlier band equal to $1024/512 \times 512$, which is $2/512$. From this table we see that at least eight inliers must be present to have a less than 5% chance (4.54% to be exact) that these inliers are the result of an accidental alignment.

The same statistical analysis has been made of the HT when it is used for the

No. of Inliers l	*Prob. of accidental alignment*			
	B = 1/512	B = 2/512	B = 3/512	B = 4/512
1	.8584	.9800	.9971	.9996
2	.5814	.9017	.9804	.9965
3	.3106	.7484	.9314	.9843
4	.1343	.5483	.8355	.9525
5	.0483	.3528	.6948	.8900
6	.0148	.2000	.5300	.7920
7	.0039	.1007	.3692	.6641
8	.0009	.0454	.2349	.5210
9	.0001	.0184	.1368	.3811
10	.0000	.0068	.0732	.2598

Table 2.3: The probability of l inliers (or more) appearing accidentally for various values of B with N equals 1000.

purpose of pose determination [Grimson and Huttenlocher, 1990]. The conclusion is that the HT will not function when the percentage of outliers is very large, and this is said to be a significant drawback of the method. In fact, our analysis shows that this problem is not particular to the HT, but is inherent in any extraction process. If the number of inliers is below a certain threshold the result is likely to have occurred because of an accidental alignment of points. The significance threshold increases with band size since for larger sized inlier bands there are likely to be more accidental inliers. This is clearly seen in table 2.3 which shows a rapidly increasing probability of accidental alignment for larger inlier bands (increasing B). However, it still may be the case that a result with more than 50% inliers is significant. This is demonstrated in Figure 2.7, in which lines are extracted from two-dimensional geometric data. The original line was created synthetically and the noise points were added uniformly across the entire image. In part (a) of this figure, the correct line has been found even when the outliers constitute 80% of the total number of points. Part (b) of the same figure shows that if enough noise points are added, then eventually breakdown occurs, and the original line is not found. However, it took far more than 50% outliers for this to occur.

Like all statistical models the usefulness of computing the significance threshold in this way depends on whether the data meets the assumptions, which are that of

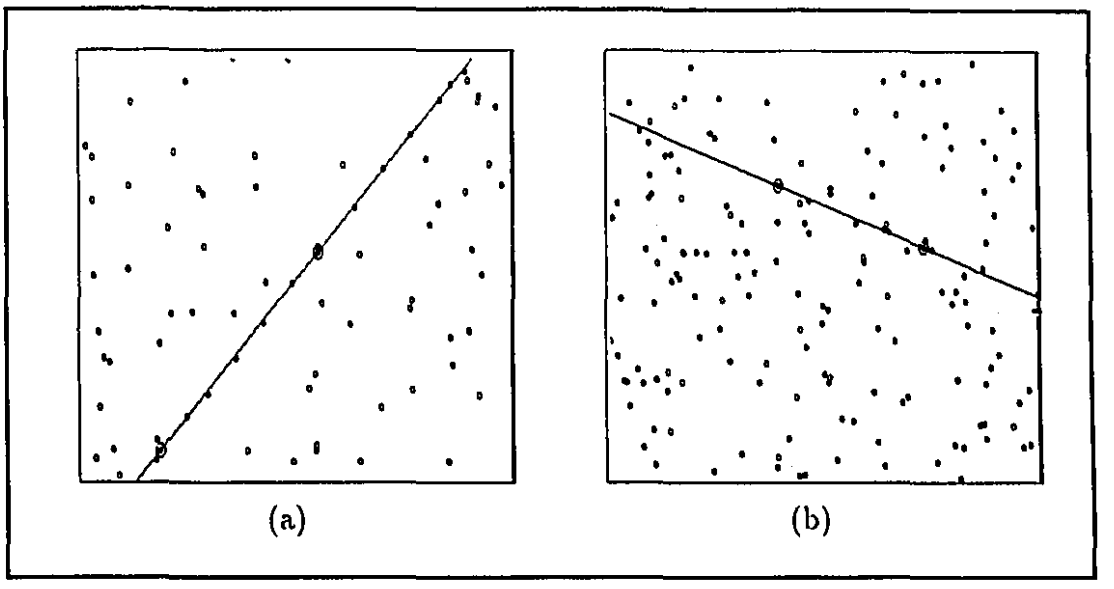

Figure 2.7: Extracting lines from noisy data (a) 80% outliers (b) 95% outliers.

a uniform distribution. In any realistic situation this statistical model does not hold perfectly and the actual setting of the significance threshold is usually determined empirically. It will always be a tradeoff between the number of false alarms and the ability to find geometric primitives. What we have shown, is that an extraction result may be statistically significant even with fewer than 50% outliers. Thus while breakdown is an intuitively appealing concept, it obscures the important point that the user of any robust method is still responsible for determining whether the primitive returned by the extraction algorithm is significant.

The setting of the significance threshold also has another important use for a fixed-band cost function. This is to provide an estimate of ϵ, the probability of a single randomly drawn geometric data point being an inlier. This issue was discussed briefly in Section 2.4.1, in which ϵ is used to compute K, the required number of randomly sampled minimal subsets. For a fixed-band cost function the significance threshold (which we have called Y), is the minimum number of inliers that must be in a primitive in order for it to be accepted as valid. If there are N geometric data points, then the probability of a randomly drawn point being an inlier is Y/N. This is the value of ϵ, from which K, the maximum number of randomly sampled minimal

subsets necessary for successful extraction, can be computed. If K random subsets are chosen and no primitive is found with at least Y points, then we can be confident that there is no such primitive (see Section 2.4.1). Since there is always an implied significance threshold for any extraction algorithm, using this threshold to estimate ϵ is a principled way of computing K, the required number of randomly sampled minimal subsets.

2.6 Summary

In this chapter we have defined the concept of primitive extraction. We have shown that the problem of primitive extraction is a generalization of fitting. In extraction the goal is to find the best subset of the geometric data points that belong to a given type of geometric primitive. We showed that this is equivalent to finding the global optimum of a cost function which usually has many local optima. This optimization model is applied to a number of extraction algorithms. The minimal subset principle is then defined, and is the basis of our minimal subset random sampling algorithm for extraction. This algorithm is guaranteed to find the global optimum for perfectly accurate data if all possible random samples are evaluated. The number of random samples necessary for successful extraction is usually considerably less than the maximum. Some ways are described to decrease the execution time of this process without using parallel hardware. The issue of the robustness of an extraction algorithm is discussed in detail.

Chapter 3 Elimination Theory for Solving Conversion Equations

This chapter discusses the use of elimination theory to convert between the minimal subset points and the parameter vector defining the geometric primitive. By using elimination theory the equations mapping a minimal subset to a parameter vector are produced in closed form, which makes them very efficient to evaluate. This is an important advantage of a symbolic approach over a numerical approach [Buchberger, 1989]. The ability to produce this mapping, and to evaluate it efficiently, is essential to the operation of our primitive extraction algorithm.

The general issue of how geometric primitives are represented is an important one, and is discussed in the beginning of the chapter. We then give an overview of the field of elimination theory, and describe the particular approach we have chosen. This is called the Gröbner bases approach [Buchberger, 1985], and we show why we prefer it over resultants, the other main alternative. The first use we make of elimination theory is to convert from the parametric to the implicit form of a geometric primitive. This is followed by a number of applications of elimination theory using examples from Chapters Two and Four.

3.1 Representing Geometric Primitives

In model-based vision the assumption is that each object is described by a set of geometric primitives. This description is normally contained in a CAD database. The geometric primitives in such a database are usually defined in parametric form. However, our algorithms require that these primitives be in implicit form. The parametric form is also sometimes called the explicit form, since each coordinate is written explicitly as a function of a number of parameters. A 2D curve is defined in parametric form as a set of points S in the following fashion:

$$S = \{(x, y) : x = f(t), y = g(t), t \in R\} \tag{3.1}$$

Similarly a 3D curve is defined as:

$$S = \{(x, y, z) : x = f(t), y = g(t), z = h(t), t \in R\} \tag{3.2}$$

A 3D surface is defined as:

$$S = \{(x, y, z) : x = f(u, v), y = g(u, v), z = h(u, v), (u, v) \in R^2\} \tag{3.3}$$

In this notation the domain of the parameters t, u, and v is R, the set of real numbers. A curve is parameterized by a single variable (t), while a surface requires two (u, v). The parametric form is ideal for generating and manipulating the shape of a curve or surface. The points on the curve or surface can be generated by varying the parameter(s) through a range of real values. The curve or surface can be manipulated by rewriting the parametric form in terms of control points. Then the shape of the curve or surface can be easily and naturally modified by moving these control points. Since generation and manipulation are the most common activities in CAD systems, the parametric form is very useful [Foley and Dam, 1982, Faux and Pratt, 1979].

A curve or surface in implicit form is defined as the set of points which are the zeros of a function(s). A 2D curve in implicit form is a set of points S defined in the following fashion:

$$S = \{(x, y) : f(x, y) = 0\} \tag{3.4}$$

Similarly a 3D surface is defined as:

$$S = \{(x, y, z) : f(x, y, z) = 0\} \tag{3.5}$$

A 3D curve is defined as the intersection of two 3D surfaces as follows:

$$S = \{(x, y, z) : f(x, y, z) = 0,\ g(x, y, z) = 0\} \tag{3.6}$$

The notation we use for a single implicit function is $f(\overline{p}; \overline{a})$. Here $\overline{p}$ is the datum point, and $\overline{a}$ defines the parameter vector for this particular primitive. This parameter vector

is not the same as the parameters used to generate a curve or surface in parametric form. For the implicit form, different curves or surfaces are defined by changing the value of the parameter vector $\overline{a}$. As an example of this notation, a 2D line is defined implicitly by the equation $a_0 + a_1 x + a_2 y = 0$, where the parameter vector $\overline{a}$ is (a_0, a_1, a_2) and the datum point $\overline{p}$ is (x, y). The implicit form naturally divides space into three regions: $f > 0$, the points on one side of the curve or surface; $f < 0$, the points on the other side of the curve or surface; and $f = 0$, the points on the curve or surface. As we have shown in Chapter Two it is easy to find an approximation to the closest distance of a point to a curve or surface when the implicit form is available.

So far nothing has been said about the particular types of functions that are used in the parametric or the implicit form. In this chapter we will discuss in detail the situation where the functions in the parametric or implicit form are polynomials, or ratios of polynomials. A polynomial function in a single unknown x is written as $a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$. While restricting these functions to polynomial form may seem significant, the polynomial representation is widely used and is very general. In most CAD systems the basic parametric form is a rational polynomial. This is because the different parametric surfaces such as the Bézier surface, Coons patches, and B-Spline surfaces, and Nurbs can all be rewritten in the form of a rational polynomial [Besl, 1988, Piegl, 1991]. For a curve, each coordinate is written in the form $a(t)/b(t)$, where a and b are polynomial expressions in a parameter t. Similarly, for a surface, each coordinate is written in the form of $a(u, v)/b(u, v)$, where a and b are polynomial in the parameters u and v. This representation is general enough to represent a wide variety of curves and surfaces.

It is possible to use elimination theory to convert any parametric form defined as a rational polynomial to its equivalent implicit form [Sederberg and Anderson, 1984]. For model-based vision, the assumption is that each object is defined by a set of geometric primitives in a CAD database. However, as we have stated these primitives are in parametric form, while our algorithms require the implicit form. Therefore converting from parametric to implicit form is the first application of elimination theory that we will demonstrate.

3.2 Essentials of Elimination Theory

Elimination theory is concerned with finding symbolic solutions to systems of equations. It concentrates on finding the solution of systems of algebraic equations, where an algebraic equation is one where the implicit function is a polynomial. However, elimination theory also studies the general requirements for finding the solution of any system of equations. While the basis of elimination theory is a century old, its modern revival is due to recent discoveries [Buchberger, 1985, Canny, 1987]. These new approaches have been little used outside of their field, but this is changing as they become better known [Kriegman and Ponce, 1990]. In the previous discussion on the representation of geometric primitives we considered a single equation of the form $f(\overline{p};\overline{a})$. In this notation $\overline{p}$ is a datum point, and here we will assume that it is an m dimensional datum point which makes $\overline{p} = (x_1, \ldots, x_m)$. In this discussion we will assume that f is a polynomial. Thus, if m is two, then the unknowns are x_1 and x_2, and the equation of f is equal to $\sum_{i,j} a_{ij} {x_1}^i {x_2}^j = 0$. Similarly, when m is three the unknowns are x_1, x_2 and x_3, and f equals $\sum_{i,j,k} a_{ijk} {x_1}^i {x_2}^j {x_3}^k = 0$. Each of the individual products of unknowns in these sums is called a term. The degree of a term is the sum of the degrees of the individual unknowns in that term. For example the term $a_{1,3,2} {x_1}^1 {x_2}^3 {x_3}^2$ has a degree of six, since this is the sum of the powers of the unknowns x_1, x_2 and x_3. The degree of an equation is that of the term with the highest degree.

A system of equations is simply a list of individual equations. For an algebraic system the total degree of the system is the product of the individual degrees of each equation. Consider the following system of n algebraic equations in m unknowns:

$$\begin{aligned} f_1(x_1, \ldots, x_m) &= 0 \\ f_2(x_1, \ldots, x_m) &= 0 \\ &\vdots \\ f_n(x_1, \ldots, x_m) &= 0 \end{aligned} \tag{3.7}$$

Such an algebraic system either has no solutions, an infinite number of solutions, or a finite number of solutions. If the number of solutions is finite, then from the theorem of Bezout we know that the number of unique solutions must be less than or equal to the total degree of the system [Hoffman, 1989].

Elimination theory can be used to find a symbolic solution to a system of algebraic equations. In a symbolic solution a closed form expression exists for each of the unknown variables in terms of the known variables. Traditionally numerical methods, such a Newton-Raphson, have been used to solve such systems of equations. However, in order to achieve convergence these methods require an initial estimate which must be close to the final solution. Another common numerical approach for solving an algebraic system are homotopies [Morgan, 1987, Renegar, 1987]. Even though homotopies find all possible solutions, they are computationally costly and have a significant number of operating parameters. By contrast, once a symbolic solution is generated, it is a very efficient way to find the solution. As its name implies elimination theory produces a symbolic solution by repeatedly combining equations to eliminate unknowns. The final result is an algebraic equation in a single unknown. If the degree of this final equation is four or less direct solutions exist; if not the solutions can be found by fast numerical methods [Hook and McAree, 1990] (see Section 3.5). The solution to the final algebraic equation is back-substituted into the previous algebraic equations. All the unknowns are solved for in this recursive fashion.

The traditional elimination theory approach uses resultants to eliminate unknown variables. A resultant operation eliminates a single variable from two equations by creating a new equation which is a logical consequence of the original equations. The following example illustrates how elimination theory using resultants operates. The goal is to find the point of intersection between a line and a circle. The equation of a line defined implicitly is $a_o + a_1 x + a_2 y = 0$ and the equation of a circle of radius r and center (x_c, y_c) defined implicitly is $(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$. A single application of the resultant operator can be used to eliminate either y or x from these two equations. The resulting equation defines the point of intersection between the line and the circle. We choose to eliminate y, and the equation produced by the

resultant operator is the following:

$$(a_2^2 + a_1^2)x^2 + (2a_0a_1 + 2a_1a_2y_c - 2a_2^2x_c)x + a_2^2x_c^2 + a_0^2 + a_2^2y_c^2 - a_2^2r^2 + 2a_0a_2y_c = 0 \tag{3.8}$$

This is an algebraic equation of degree two in a single unknown x. It is implied by the two original equations in the sense that it must be satisfied if the value of x satisfies both the original two equations. Since the equation is a polynomial of degree two in a single unknown, it is a quadratic equation. Therefore a formula is available for finding x, and by substituting each value of x into either of the two original equations, y can also be found. This (x, y) pair is the intersection point between the line $a_o + a_1x + a_2y = 0$, and the circle $(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$.

The difficulty with the resultant approach is that it becomes impractical for a system with more than two or three equations. The reason is that when using resultants the final algebraic equation always has a degree equal to the total degree of the system. The total degree is the product of the degrees of each of the equations in the system. The theorem of Bezout says that the number of distinct solutions to an algebraic system is less than or equal to its total degree. However, because there are multiple roots, it is often the case that the number of distinct solutions is considerably less than the total degree. The resultant approach does not take this into account. This means that the degree of the final equation is often unnecessarily large, since it contains multiple roots. This problem is inherent in all resultant based methods, including multi-variable resultants [Canny, 1987]. What we would like is an elimination method which takes this into account to produce a polynomial in a single unknown of the lowest possible degree. There is such a method, and it is called the Gröbner bases approach.

3.2.1 Gröbner bases

The Gröbner bases method is a new and powerful approach to solving an algebraic systems of equations [Buchberger, 1985]. The method has implications beyond equation solving, and has been applied to a wide variety of difficult and important problems

[Hoffman, 1989]. In the general case computing the Gröbner bases is very computationally intensive. However, for certain systems of algebraic equations more efficient versions of the algorithm exist. Such a situation occurs when the algebraic system is known to have a finite number of solutions. The system of equations produced by the minimal subset algorithms has this characteristic, and can thus be solved more easily than other algebraic systems. While the underlying theory of the Gröbner bases method is complex, the output of the algorithm is easy to understand. When applied to a system of algebraic equations the algorithm is guaranteed to eventually terminate with a Gröbner basis, which is another system of algebraic equations. When the original system has at least one solution then the number of equations in this new system is the same as in the original. It has been proven that any solution to the original system is also a solution of the Gröbner basis system, and vice-versa.

However, the Gröbner basis system can be forced to be in triangular form. In this form the last equation has a single unknown, the second last has two unknowns, one of which is the unknown in the last equation, and so on. For a triangular system of equations back substitution can be used to find the solutions to all the roots to any desired level of accuracy. Since the last equation has a single unknown it can be solved directly using a formula, or by fast numerical methods (see Section 3.5). When the solution for the last equation is substituted into the second to last equation, then this equation in turn only has a single unknown, and it can also be solved. This process is called back substitution, and can be applied repeatedly to find all the solutions for the system. We use the example of finding the intersection between a line and a circle to demonstrate this method. The resulting Gröbner basis consists of the following two equations. Here the first member of the basis is the same as the first equation in the original system, but this is not always the case.

$$a_0 + a_1 x + a_2 y = 0 \tag{3.9}$$

$$a_2^2 x_c^2 + a_2^2 y_c^2 - a_2^2 r^2 + 2a_0 a_2 y_c + a_0^2 + (2a_0 a_1 - 2a_2^2 x_c + 2a_1 a_2 y_c)x + (a_2^2 + a_1^2)x^2 = 0 \tag{3.10}$$

The last equation of the two has only one unknown, which is x. It is identical to the equation produced by the resultant approach, but this is not always the case.

Because it produces equations of the lowest possible degree, the Gröbner bases approach is potentially able to solve much larger systems than the resultant method. The difficulty is that the computational time necessary to produce the basis is very large. The running time of the Gröbner bases algorithm is doubly exponential in the total degree of the system. The algorithm has been proven to always terminate, but there is no way of knowing how long this will take [Buchberger, 1985]. However, the symbolic solution need only be generated once, and can thereafter be used to efficiently solve the system. Considerable research is being done on producing faster versions of the Gröbner bases method [Gatermann, 1990, Hoffman, 1989].

3.3 Conversion Examples

The first use of elimination theory is to convert from the parametric to the implicit form of a curve or surface. In CAD systems geometric primitives are represented in parametric form. For our algorithms we need these equations in implicit form. The basic parametric form used in many CAD systems is the rational polynomial [Besl, 1988]. When a curve is defined in this way each coordinate is written as $a(t)/b(t)$, where a and b are polynomials in the parameter t. Similarly, when defining a surface, each coordinate is written in the form of $a(u,v)/b(u,v)$, where a and b are polynomials in the parameters u and v. Strictly speaking curves and surfaces in parametric form are not directly represented as rational polynomials in many CAD systems, but are stored in B-Spline, Bézier , or Nurb form [Farin, 1993]. However, it is possible to convert these descriptions into the rational polynomial form by using numerical methods described in the literature [Farin, 1993].

The conversion of a geometric primitive given in parametric form to the implicit form is demonstrated in the following example. The rational polynomial representation of a circle with the origin at the center and radius r is $x = r((1-t^2)/(1+t^2))$

and $y = r((2t)/(1+t^2))$. This implies that the following two equations hold:

$$x - r((1-t^2)/(1+t^2)) = 0 \qquad (3.11)$$
$$y - r((2t)/(1+t^2)) = 0$$

To convert from the parametric to implicit form it is only necessary to use elimination theory to remove the variable t from these two equations. This can be done by using either resultants or the Gröbner bases method. The result of this elimination process is the equation of the circle in implicit form which is $x^2+y^2 = r^2$. This approach can also be used to convert a rational polynomial description of a surface to its equivalent implicit form. For a surface there are three parametric equations, instead of two, so two variables must be eliminated. The ability to convert from parametric to implicit form is important because our algorithm requires the implicit form, and the geometric models of objects in most CAD databases are defined parametrically.

A difficulty with this conversion process is that the resulting algebraic equations are sometimes of a very high degree. For a 2D curve in the x, y plane defined parametrically in terms of a degree n rational polynomial in t, the implicit form is a degree n algebraic equation in x and y. Similarly, for a surface defined parametrically as a degree m rational polynomial in u and v, the resulting implicit surface is an algebraic equation of degree $2m^2$ in x, y and z. The most common surfaces defined parametrically have degree 3, and these are equivalent to degree 18 polynomials in implicit form. Such a high order polynomial has very many terms (over 1000). This makes the number of points in the minimal subset for this surface far too large for our algorithm.

The high degree of the associated implicit surfaces for various parametric forms leads us to believe that such geometric primitives have more degrees of freedom than are necessary. This has led to research in the use of certain parametric forms that convert to lower order implicit functions, such as the Steiner surface [Sederberg and Anderson, 1985]. There has also been interest in the direct use of lower order implicit polynomial functions as design surfaces, instead of using the

parametric forms [Bajaj and Ihm, 1992, Sederberg, 1985]. The advantages of having both the parametric and implicit form of a surface available in a CAD system are significant. For this reason the description of the patches defining objects may in the future be in parametric forms that can be converted to low order implicit polynomials. This will make our algorithm more practical for extracting the surfaces defining free-form objects found in many CAD systems.

3.4 Extraction Examples

In the primitive extraction algorithm it is necessary to convert from a minimal subset to the parameter vector that defines a primitive. Each of the R points in a minimal subset must be on the curve or surface. Therefore the system of R equations $f(\overline{p}_1; \overline{a}) = 0$ to $f(\overline{p}_R, \overline{a}) = 0$ created by replacing the variable $\overline{p}$ by $\overline{p_1}$ to $\overline{p}_R$ in f must hold. The unknown in this system is the parameter vector $\overline{a}$, and the solution to this system defines the mapping between a minimal subset and a parameter vector.

We will consider the solution of this system for two different types of implicit functions. In the first case, f is restricted to be a linear combination of a possibly nonlinear set of basis functions. In the second case, f is a polynomial, which cannot necessarily be written as a linear function. These two cases cover a wide variety of curves and surfaces. It should be noted that they are not mutually exclusive, and some geometric primitives (such as a circle) can be written in both forms.

3.4.1 Linear Combination of Basis Functions

We first consider the case where f is restricted to be a linear combination of a possibly nonlinear set of basis functions. If there are R basis functions labelled b_1 to b_R then $f(\overline{p}) = a_0 + \sum_{i=1}^{i=R} a_i b_i(\overline{p})$. In this notation $b_i(\overline{p})$ is basis function b_i evaluated at $\overline{p}$, and a_i is the coefficient of this basis function. The dimension of the parameter vector $\overline{a} = (a_0, \ldots, a_R)$ is $R+1$, where the minimal subset is of size R. Thus the size of the minimal subset always equals the number of basis functions. Such a representation is general enough to describe a wide variety of curves and surfaces. Table 3.1 gives

Shape	*Basis Functions*
line	x, y
circle	$x, y, x^2 + y^2$
conic	x, y, x^2, xy, y^2
cubic	$x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3$
plane	x, y, z
sphere	$x, y, z, x^2 + y^2 + z^2$
quadric	$x, y, z, x^2, y^2, z^2, xy, yz, zx$

Table 3.1: Some common shapes and their associated basis vectors.

some common shapes and their associated basis functions [Pratt, 1987].

If we substitute $\overline{p}_1, \overline{p}_2, \ldots \overline{p}_R$ for $\overline{p}$ in f we have R equations. However, $f(\overline{p}; \overline{a})$ must hold for any point $\overline{p}$ on f, and if this equation is added to the set we obtain the following linear system:

$$\begin{bmatrix} 1 & b_1(\overline{p}) & \ldots & b_R(\overline{p}) \\ 1 & b_1(\overline{p}_1) & \ldots & b_R(\overline{p}_1) \\ \vdots & \ldots & \vdots & \\ 1 & b_1(\overline{p}_R) & \ldots & b_R(\overline{p}_R) \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_R \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.12}$$

Even though the basis functions are not necessarily linear, the above system is linear in the unknowns $a_0, \ldots, a_R$. The fundamental theorem of elimination says that a non-trivial solution for a linear system exists only if the determinant of the associated matrix equals zero [Macaulay, 1916]. This means that by expanding the following determinant, setting it equal to zero, and collecting terms, the unknowns can be solved for:

$$\begin{vmatrix} 1 & b_1(\overline{p}) & \ldots & b_l(\overline{p}) \\ 1 & b_1(\overline{p}_1) & \ldots & b_l(\overline{p}_1) \\ \vdots & \ldots & \vdots & \\ 1 & b_1(\overline{p}_R) & \ldots & b_l(\overline{p}_R) \end{vmatrix} \tag{3.13}$$

As an example, consider the case of a line which is described by the implicit equation $a_0 + a_1 x + a_2 y = 0$. Here the basis functions are x and y, which makes b_1 equal to x and b_2 equal to y. Two input points $\overline{p_1} = (x_1, y_1)$ and $\overline{p_2} = (x_2, y_2)$ are

necessary and sufficient to define a unique line. Expanding the following determinant and setting it to zero produces the equation of the line.

$$\begin{vmatrix} 1 & x & y \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{vmatrix} \tag{3.14}$$

The expansion of this determinant is the following equation.

$$x_2y_1 - x_1y_2 + x(y_1 - y_2) + y(x_2 - x_1) = 0 \tag{3.15}$$

from which it can be seen that $a_0 = x_2y_1 - x_1y_2$, $a_1 = (y_1 - y_2)$, and $a_2 = (x_2 - x_1)$.

This method of expanding determinants will produce a closed form solution for each element of the parameter vector $\overline{a}$ as a function of the R points in a minimal subset. This technique will work whenever f is a linear combination of basis functions, which may themselves be nonlinear. This method of computing the parameter vector was used for the line, circle, ellipse, plane, sphere and quadric extraction examples described in Chapters Two and Four.

3.4.2 Polynomial Function

We will now consider the case where f is a polynomial function, which is an important case for two reasons. The first reason is that algebraic equations describe the geometric primitives implemented in many CAD systems. In Section 3.3 we noted that when parametric surface patches defined as rational polynomials are converted to implicit form the result is an algebraic equation [Sederberg and Anderson, 1984]. In addition, it has been shown that the outlines of objects modelled by such surface patches are also curves which are defined by algebraic equations [Kriegman and Ponce, 1990]. Thus the ability to extract geometric primitives defined by algebraic equations is important if our approach is to apply to a wide variety of geometric primitives.

Some of the curves defined in the previous section as a linear combination of basis vectors can also be written as polynomials. When there are no constraints on these

curves, then the linear form is the more useful. However, when there are constraints these equations are usually not linear in the parameter vector, and must be written in their polynomial form. It is important to be able to solve such algebraic systems because when these constraints exist, they lower the degrees of freedom of the primitive. For example, a circle with a known radius has two degrees of freedom, versus three for an unconstrained circle, and a cylinder (which is a constrained quadric) has six degrees of freedom, verses nine for a quadric. The number of points in a minimal subset equals the degrees of freedom of the geometric primitive. In Chapter Two we showed that the number of random samples necessary to achieve successful extraction is an exponential function of the number of points in a minimal subset. Therefore reducing the size of a minimal subset enables the required number of random samples to be significantly reduced.

We will consider further the example of extracting a circle which has a known radius. An unconstrained circle can be written as a linear combination of basis functions, with these functions consisting of $x, y, x^2 + y^2$. However, if the radius is known the elements of the parameter vector $\overline{a}$ are constrained, and this linear description of the circle is no longer valid. Instead, the circle must be written as $f(x,y) = (x - C_x)^2 + (y - C_y)^2 - r^2$, where the unknowns are the coordinates of the circle center (C_x, C_y). In this case f is a polynomial but is not a linear function of the parameter vector (C_x, C_y). Therefore the previous approach to finding the parameter vector $\overline{a}$, which uses determinants will not work. To solve such algebraic systems we use the Gröbner basis method described in Section 3.2.1.

If the two minimal subset points, (x_0, y_0), and (x_1, y_1) on the constrained circle are substituted into the circle equation, then the following algebraic system is the result.

$$\begin{aligned} (x_0 - C_x)^2 + (y_0 - C_y)^2 - r^2 &= 0 \\ (x_1 - C_x)^2 + (y_1 - C_y)^2 - r^2 &= 0 \end{aligned} \tag{3.16}$$

Here the known, or given variables, are x_0, y_0, x_1, y_1 and r, since the radius is fixed.

The unknown variables are the circle center C_x and C_y. The Gröbner basis method is used to find two solutions in closed form, and both are listed in Appendix A.1.

We will now demonstrate how the Gröbner basis approach works by solving the system of equations for a number of the extraction examples that will be described in more detail in Chapter Four. In the first example we consider an ellipse where the center and one of the axes is constrained to a given value. Without loss of generality, we can assume that the center of the ellipse is at the origin. Then the equation of the ellipse is the following:

$$b^2x^2 + a^2y^2 - a^2b^2 = 0 \tag{3.17}$$

The two axes of the ellipse have length a and b respectively. If the ellipse is rotated about the origin by an angle θ, then by substituting $s_1 = \cos(\theta)$ and $s_2 = \sin(\theta)$ the following polynomial is obtained:

$$b^2(xs_1 - ys_2)^2 + a^2(xs_2 + ys_1)^2 - a^2b^2 = 0 \tag{3.18}$$

For an unconstrained ellipse the free variables are the center, the two axes, and the orientation, which gives five degrees of freedom. Then the ellipse can be represented by the conic equation listed in Table 3.1. However, here the center of the ellipse, and one of the axes (a) are known. Thus the free variables are the rotation angle (θ) and the other axis (b). This means that the constrained ellipse has two degrees of freedom, and two points are necessary and sufficient to define this curve. If the two points are (x_0, y_0) and (x_1, y_1), and they are substituted one at a time into the above equation, then the following algebraic system with unknowns s_1, s_2, and b, is the result:

$$\begin{aligned} b^2(x_0s_1 - y_0s_2)^2 + a^2(x_0s_2 + y_0s_1)^2 - a^2b^2 &= 0 \\ b^2(x_1s_1 - y_1s_2)^2 + a^2(x_1s_2 + y_1s_1)^2 - a^2b^2 &= 0 \\ s_1^2 + s_2^2 - 1 &= 0 \end{aligned} \tag{3.19}$$

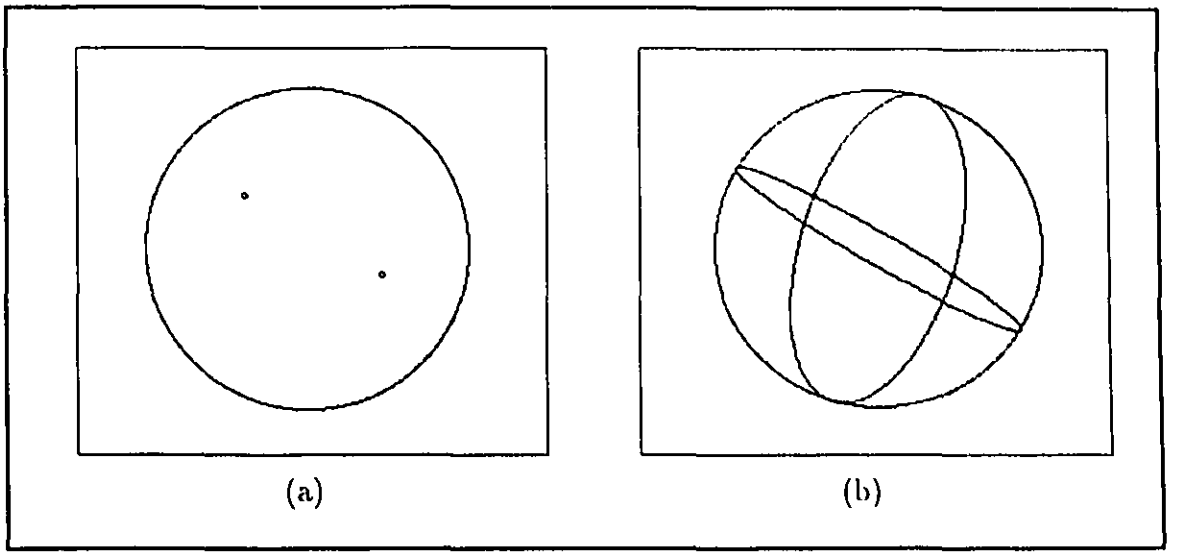

Figure 3.1: Constrained ellipses through two points (a) Circle and two points (b) Two ellipses through two points with circle center and major axis.

The last equation simply encodes the constraint that $\cos^2(\theta) + \sin^2(\theta) = 1$. There are two different solutions to the system and they are described in Appendix A.2. In Figure 3.1 part (a), a circle is shown along with two points on such a constrained ellipse. We constrain the ellipse to have as its center the circle center, and as its major axis the circle diameter. In part (b) of the figure, we draw the two ellipses produced from the solution of the above set of equations, and they both satisfy these constraints. This example demonstrates that constraints on the geometric primitive reduces the size of minimal subset, in this case from five points to two points.

Consider the case where the geometric primitive is a curve in 3D space. Such curves are defined as the intersection of two surfaces, and the resulting system of equations is rarely linear. As an example consider a circle in 3D Cartesian space. It can be described as the intersection of a sphere and a plane through the sphere's origin. This is shown in Figure 3.2, where the plane and sphere are drawn. The size of the minimal subset is three, since three points are necessary and sufficient to uniquely define this curve. The equation of a plane is $a_0 + a_1 x + a_2 y + a_3 z = 0$, and the equation of a sphere with radius r and center (x_c, y_c, z_c) is $(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$. We assume that the normal to the plane is a unit normal, which adds the equation

$a_1^2 + a_2^2 + a_3^2 = 1$ to the set. The fact that the plane passes through the center of the circle adds the equation $a_0 + a_1 x_c + a_2 y_c + a_3 z_c = 0$ to the set. Each point on the 3D circle must satisfy both the sphere and plane equations. Let us assume that the points (x_0, y_0, z_0), (x_1, y_1, z_1), and (x_2, y_2, z_2) are those points on the circle. Then the following system of equations is the result of substituting the points into each of the two equations.

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 y_0 + a_3 z_0 &= 0 \\
a_0 + a_1 x_1 + a_2 y_1 + a_3 z_1 &= 0 \\
a_0 + a_1 x_2 + a_2 y_2 + a_3 z_2 &= 0 \\
(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2 &= 0 \\
(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2 &= 0 \\
(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2 &= 0 \\
a_0 + a_1 x_c + a_2 y_c + a_3 z_c &= 0 \\
a_1^2 + a_2^2 + a_3^2 - 1 &= 0
\end{aligned} \tag{3.20}$$

This system can be solved using the Gröbner basis method, and it has exactly one solution which is listed in closed form in Appendix A.3.

We have described methods of converting from the minimal subset points to a parameter vector when either f is a linear combination of basis functions, or when f is a polynomial. While not every implicit function meets these criteria, these types of functions do cover a wide variety of commonly used shapes. The ability to handle polynomial equations is practically important because they are the basis of the descriptions in many CAD systems.

3.5 Numerical Solutions

In our extraction examples elimination theory is used to eliminate unknowns in order to produce an algebraic equation with only a single unknown. The solution of this algebraic equation must be found and substituted back into the previous equation of

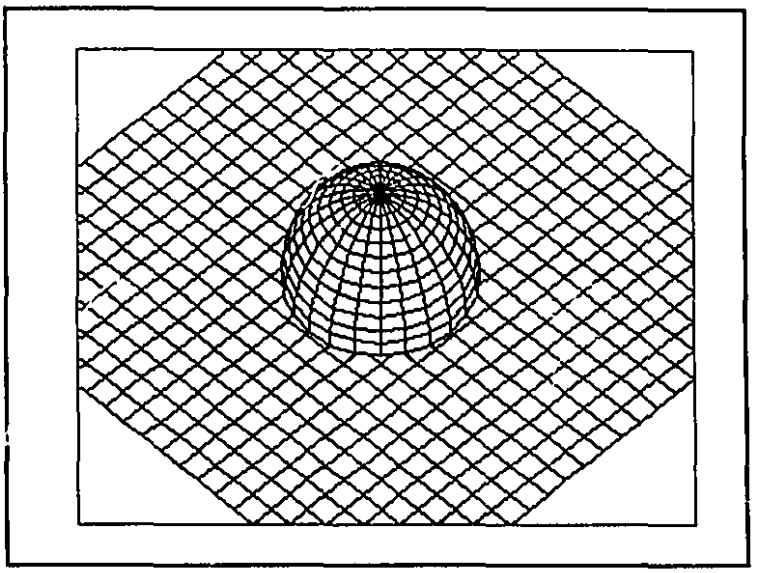

Figure 3.2: Three dimensional circle is the intersection of a plane and a sphere

the triangular Gröbner basis system. The previous equation now has only a single unknown, and the solutions for it must be found. This procedure is applied recursively until all the solutions are found to the entire algebraic system. The assumption is that an algebraic equation in a single unknown can be solved either symbolically or numerically. While correct, there are some ways of solving such an equation which are particularly well suited to our application.

If the degree of the algebraic equation with a single unknown is four or less then there are closed form solutions available. However, if the equation has a degree higher than four numerical methods must be used to find the solutions. There is a numerical approach which is particularly useful in our application. It is based on Sturm sequences, and implementations are described in the literature [Hook and McAree, 1990]. For the extraction application we are only interested in the real solutions. When the solutions have a complex component this implies that the minimal subset does not meet the required constraints. This is illustrated by our example of finding a circle with a given radius through two points. In this case a complex solution means that no circle through the two planar points exists. Thus, we want to find only the *real* solutions of an algebraic equation with a single unknown. It is also true that there are often constraints on the range of possible values of each

unknown.

Using a Sturm sequence is the preferred ways of achieving this goal. It is superior to other methods in the literature for two reasons [Press and Flannery, 1988]. First, this approach finds only the real roots, second it can deal with very ill-conditioned equations, and third it can accept constraints on the solutions. Informally, a system is ill-conditioned if a small change in the coefficients of a polynomial can dramatically change the number of real roots, and their locations. Often if a polynomial has real roots, it will be ill-conditioned. The following algebraic equation is an example of this situation:

$$W(x) = (x+1)(x+2)\ldots(x+20) = x^{20} + 210x^{19} + \ldots + 20! = 0 \qquad (3.21)$$

Clearly there are twenty real roots ranging from minus one to minus twenty to this equation. Let us change the coefficient of x^{19} slightly to produce a new algebraic equation $U(x)$ defined as follows.

$$U(x) = W(x) + 2^{-23}x^{19} = x^{20} + (210 + 2^{-23})x^{19} + \ldots + 20! = 0 \qquad (3.22)$$

It would seem that such a trivial change in a single coefficient would have little impact on the roots. However, this new equation has only ten real roots, and the imaginary parts of the other roots are fairly large [Wilkinson, 1959]. The fact that all the real roots of such an equation can be found reliably using a Sturm sequence demonstrates the robustness of this root finding method.

There is public domain software available to compute the Sturm sequence [Hook and McAree, 1990]. Once the Sturm sequence has been computed all the real roots are found by a simple bisection algorithm. The use of the Sturm sequence for finding the real roots of an algebraic equation with a single unknown makes the elimination theory approach much more practical in many more applications.

3.6 Summary

This chapter has concentrated on the application of elimination theory to our extraction algorithm. This theory is used to produce the closed from solutions to a system of equations defined using minimal subsets. The solution to such a system maps from the minimal subset points to the parameter vector defining a geometric primitive. While elimination theory is computationally intensive, it need only be applied once to produce the closed form solution. Thereafter this solution can be used repeatedly in the extraction algorithm, and is usually very efficient. The examples in this chapter were processed using the Gröbner basis routines found in Maple, a symbolic algebra package [1].

We first discussed our representation of geometric primitives and transformations. Then we gave an overview of the field of elimination theory. This was followed by a number of extraction examples that showed how elimination theory is applied. To actually solve the resulting Gröbner basis it is necessary to repeatedly find the real roots of an algebraic equation with a single unknown. We concluded by showing how the Sturm sequence method can be used to achieve this goal in a very reliable fashion.

[1] Maple is a product of the Symbolic Computation Group, Computer Science Dept., University of Waterloo, Canada

Chapter 4 Extraction Applications Using Random Sampling

Primitive extraction as we have defined it is a generalization of fitting. A single invocation finds a single geometric primitive in the geometric data. How can we use our extraction algorithm to deal with the situation in which there is more than one geometric primitive, or where the primitives are not necessarily all of the same type? This problem is closer to the traditional definition of segmentation [Levine, 1985]. By itself, our extraction algorithm is not a segmentation algorithm. However, in this chapter we will show how a number of different segmentation algorithms can be built using our extraction algorithm. Before we do this we will first discuss some deficiencies of current segmentation algorithms.

In our opinion the following are what we consider to be the most desirable characteristics of a segmentation algorithm:

- The algorithm should have no restrictions on the type of geometric primitives, and should be able to find both curves and surfaces.
- The algorithm should be able to operate on both single view and multiple view data.
- The algorithm should operate on both sparse and dense geometric data.
- The algorithm should be able to handle geometric data of varying noise levels produced by different geometric sensors.
- The algorithm should have a formal mathematical basis.
- The algorithm should be simple thereby permitting others to implement it easily and to verify the experimental results.
- The algorithm should have a small number of operating parameters.

In light of these criteria, the existing segmentation algorithms in the literature each have a number of drawbacks [Besl, 1988, Hoffman and Jain, 1987, Yokoya and Levine, 1989, Boulanger and Godin, 1992]. The most serious is the requirement that the geometric data be in standard image format. This means that the data must be presented in the form of a dense set of points described by a graph function as $z = f(x, y)$. We have defined geometric data in a very general way, in which the data are not necessarily available as a graph function. This is because we want to accommodate a wide variety of different methods of producing geometric data.

We believe that by using our extraction algorithm it is possible to create various segmentation algorithms which satisfy the above criteria better than other methods described in the literature. In thi chapter we will describe how this is accomplished. This will be followed by a number of experimental results. They will demonstrate the extraction of geometric primitives for a variety of geometric data. Finally, we will compare our method in detail with the robust statistics approaches, the minimal length encoding schemes, and the Hough transform (HT).

4.1 Multiple and Different Primitives

The first case we consider is that in which there are multiple primitives to be extracted which are all of the same type. The second case is a generalization of the first, in which there are multiple primitives that are not necessarily all of the same type. We will consider the two cases separately.

If there is more than one geometric primitive of the same type to be extracted, the obvious approach is to apply the random sampling extraction algorithm repeatedly. Following the first application, the inliers are removed, and the next application has as input the remaining geometric data points. This process continues as long as the extraction is successful, where success means that the value of the cost function of the best primitive is significant. However, if the number of geometric primitives or the size of a minimal subset is large, then an excessive number of random samples may

be necessary. In order to decrease this number, we propose to divide the geometric data into locally connected subsets. Each such connected subset is likely to consist of a small number of curves or surfaces. Therefore performing random sampling on each connected subset instead of all the geometric data should require fewer samples for successful extraction. The disadvantage of using local connectivity is that the optimal solution may not be found. However, the advantage is that when there are a significant number of geometric primitives to extract the number of random samples is greatly reduced.

The particular connectivity algorithm used depends on the type of geometric data. For two-dimensional geometric data consisting of edge points from an intensity image, standard algorithms can quickly produce all of the connected subsets [Levine, 1985]. If the geometric data are dense range data taken from a single view we find the largest connected region bounded by edge points. For dense range data, an edge point is a discontinuity in depth or orientation, and these points can easily be found by local operators [Godin and Levine, 1989, Boulanger *et al.*, 1990]. Each of these two approaches to using local connectivity is illustrated in Section 4.2. The actual data produced by a connectivity algorithm often depends dramatically on the settings of certain parameters (such as whether the connectivity is four-or eight-connected). However, because of the underlying robustness of the extraction algorithm, ultimately the same geometric primitives are likely to be extracted even when the parameters of the connectivity algorithm vary. An example of this will be shown in Section 4.2.

The following pseudo-code describes the modified extraction algorithm for multiple primitives of the same type. The input is the definition of the geometric primitive to be extracted along with the geometric data.

While there is sufficient geometric data
Do
1. *Find the largest connected subset of the geometric data.*
2. *Extract all the geometric primitives from this subset.*
3. *Save the associated parameter vectors and inliers of the extracted primitives.*
4. *Remove the inliers from the geometric data.*

Enddo

A more complex situation exists when there are not only multiple primitives, but

also different types of primitives. For example, the goal may be to extract not just lines, but also circles and ellipses. One solution is to search for all the primitives of a certain type, and then when no more exist, go to the next type. For example, when extracting lines, circles and ellipses, first extract all the lines, then when no more lines are found (according to the significance threshold) extract all the circles, etc. The problem with this approach is that if the significance threshold is too low then a complex primitive (such as a circle with a large radius) may be mistaken for simpler one (i.e. a line).

An alternative is to attempt the simultaneous extraction of each different type of primitive, and then accept only the best one. This is the approach we have taken. In terms of our previous example this would mean applying the extraction algorithm for lines, circles and ellipses simultaneously on the same geometric data. The primitive with the most inliers would be the one returned by the algorithm, as long as it was deemed to be significant. If the number of inliers for two different primitives is equal or close to equal, then the simplest primitive is chosen (eg. a line is simpler than a circle). This procedure is very similar to that used in the minimal length encoding schemes [Darrell *et al.*, 1990, Pednault, 1989], and we will compare the two approaches in more detail in Section 4.3.

4.2 Experimental Results

In this section we present a number of examples of primitive extraction for various kinds of geometric data. The two-dimensional examples show the extraction of lines, circles and ellipses. The three-dimensional examples show the extraction of planes and quadrics from range data provided by a laser rangefinder [Rioux, 1985]. All examples were processed using a fixed-band cost function whose size was determined from a study of the resolution of the geometric sensor data. In some cases there is more than one geometric primitive to be extracted, and in other cases there are different types of primitives to be extracted.

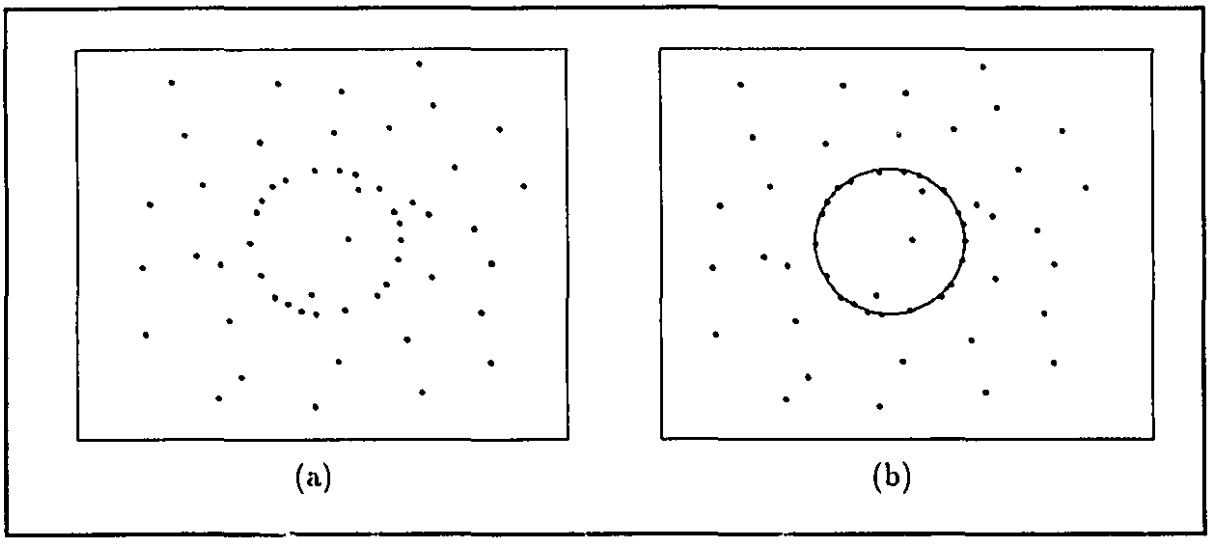

Figure 4.1: Extracting a circle (a) Initial data points (b) Extracted circle.

4.2.1 Two-Dimensional Data

The first set of examples will demonstrate the extraction of lines, circles and ellipses from two-dimensional geometric data. For some examples the data was created synthetically, and for others the geometric data were obtained by processing intensity images.

The first examples show the extraction of a single circle and ellipse from a set of points in the plane. The results are shown in Figures 4.1 and 4.2. In both figures part (a) shows the data points and part (b) shows the extracted primitive. In both cases the number of random samples used to the extract the primitive was seventy. This value of K, the number of random samples, was computed using the formula discussed in the Chapter Two. Note that in all the examples so far the percentage of inliers (points on the line, circle or ellipse) is less than 50% of the total number of points.

In Figure 4.3 we show the extraction of a number of circles from synthetic data in the plane. Part (a) of the figure shows the data points and parts (b) and (c) show the extracted circles. A fixed-band scoring function was used with a band size of ten pixels and with seventy random samples. Part (b) shows the extraction results without gap checking, and part (c) shows the extraction results with gap checking.

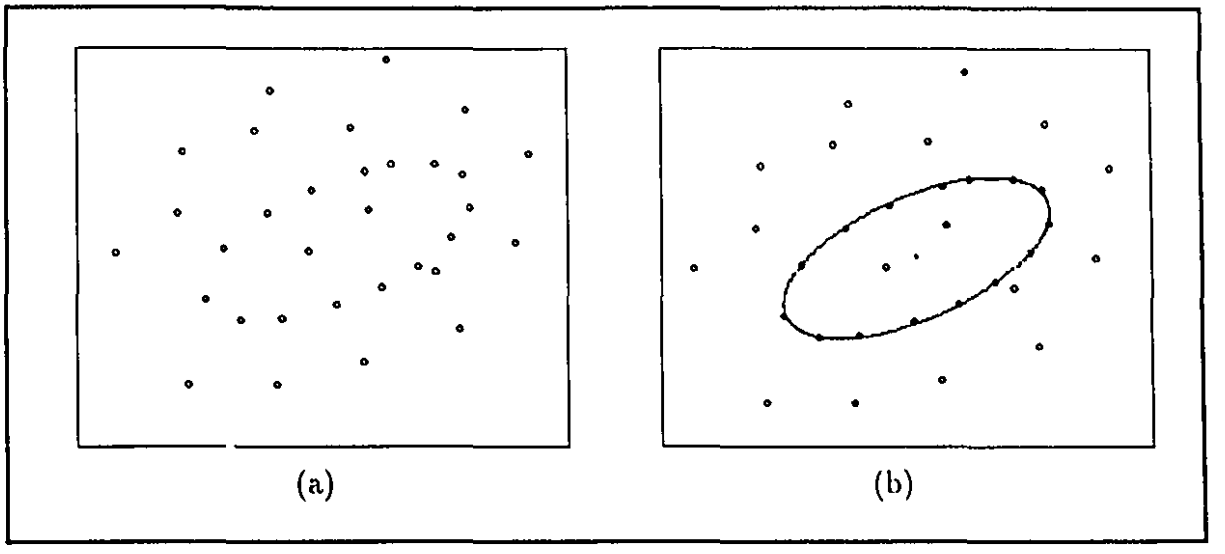

Figure 4.2: Extracting an ellipse (a) Initial data points (b) Extracted ellipse.

Gap checking is an extra parameter of the fixed-band scoring function for a two-dimensional curve which can be enabled or disabled. When enabled the curve defined by a minimal subset is scored using only the connected inliers. This means that the longest sequence of points on the curve that are within a certain distance of each other are counted, and this distance is called the gap value. This value is a parameter which must be set by the user. By contrast, the usual type of scoring is non-connected, which means that all the inliers are considered to be on the same curve, regardless of their separation. The effect of gap checking can be seen in part (c), where three circles are extracted in the place of the single circle extracted in part (b). The single circle was extracted without gap checking because three disconnected parts of the other circles made up a large circle, which had many points, and thus a good cost function value. However, from the figure it is clear that there are actually three distinct circles, and not one large circle. When gap checking is used the correct result is obtained since the gap between the disconnected portions of the larger circle is too large for them to be considered as part of the same circle. Gap checking may or may not be useful depending on the application. While we have demonstrated gap checking for circle extraction it can also be added to the extraction algorithm for other curves, such as lines and ellipses.

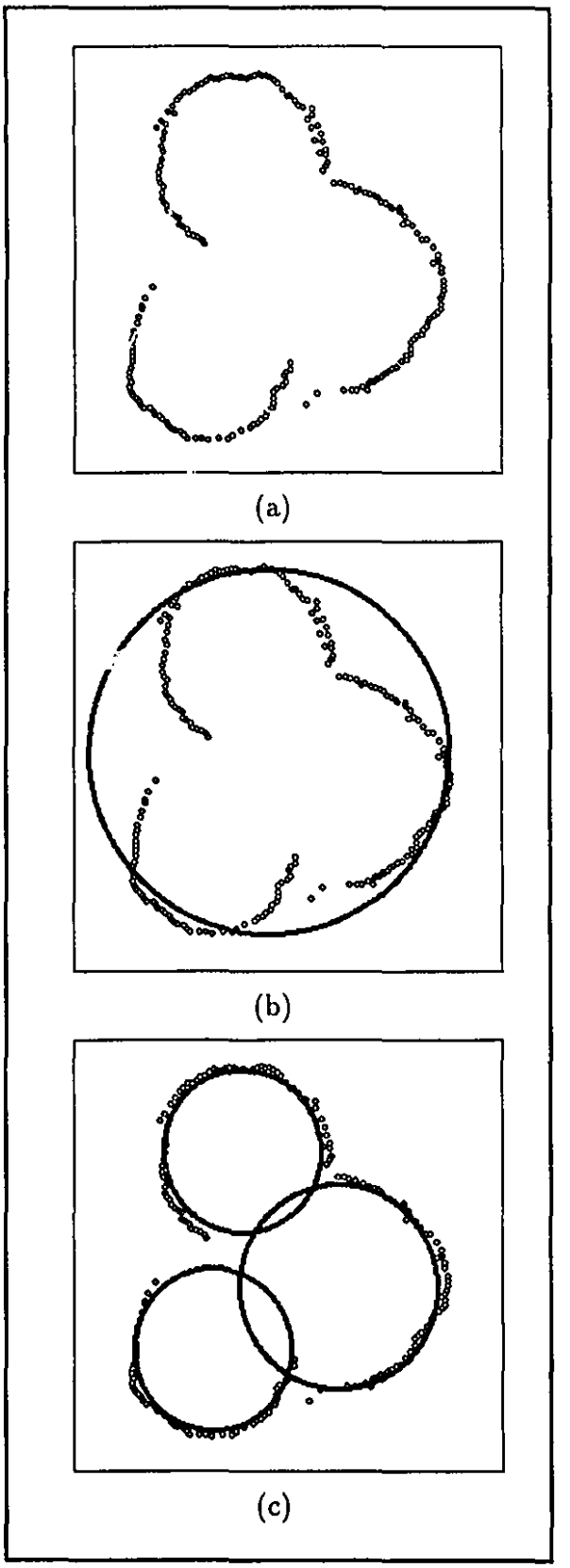

Figure 4.3: Multiple circle extraction (a) initial data points (b) Extracted circle without gap checking (c) Extracted circles with gap checking

In the fourth example ellipses and circles are extracted from an intensity image of a ball of about 50 pixels size. The outline of the ball is a circle, and a number of great circles painted on the ball projects to a number of ellipses. The motivation for this example comes from a tracking application. The objective is to track the position and orientation of the ball from image to image. This can be done by finding the poles of the ball, which are the intersections of the great circles. Since the poles completely define the position and orientation of the ball, this information is sufficient for tracking purposes. The input consists of the edge points created from the intensity image by standard edge detectors. The small size of the ball, the sparseness of the edge points, and the closeness of the ellipses makes this extraction example particularly difficult.

The following two-step process extracts the ellipses from the geometric data. First, the circle which defines the outline of the ball is extracted; then using information from the circle the ellipses are extracted. The centers of the ellipses are constrained to be at the center of the ball with their major axis equal to the radius of the ball. This means that once the circle defining the outline of the ball is extracted there are only two degrees of freedom for each ellipse (as opposed to five for a general ellipse). This makes the size of the minimal subset R two, instead of five. The procedure used to create the parameter vector of the ellipse through two randomly sampled points was discussed in detail in Chapter Three. Part (a) of Figure 4.4 shows the initial image, part (b) shows the initial edge pixels, part (c) shows the extracted ellipse and circles, while part (d) shows the extracted ellipses and circle superimposed on the image data. As can be seen from the figure, the correct ellipses have been found, and from these it is a simple matter to find the poles of the ball.

The fifth example shows the extraction of circles from edge data. Because of the relatively large number of circles, the random sampling process is limited to locally connected segments. As described in Section 4.1 this enables the circles to be extracted with fewer random samples than if all the geometric data were used as input. The results are shown in Figure 4.5. Part (a) shows the initial image, part (b) shows the edge points, part (c) shows the extracted circles, and part (d) shows the extracted circles superimposed on the edge points. Again, as can be seen from the

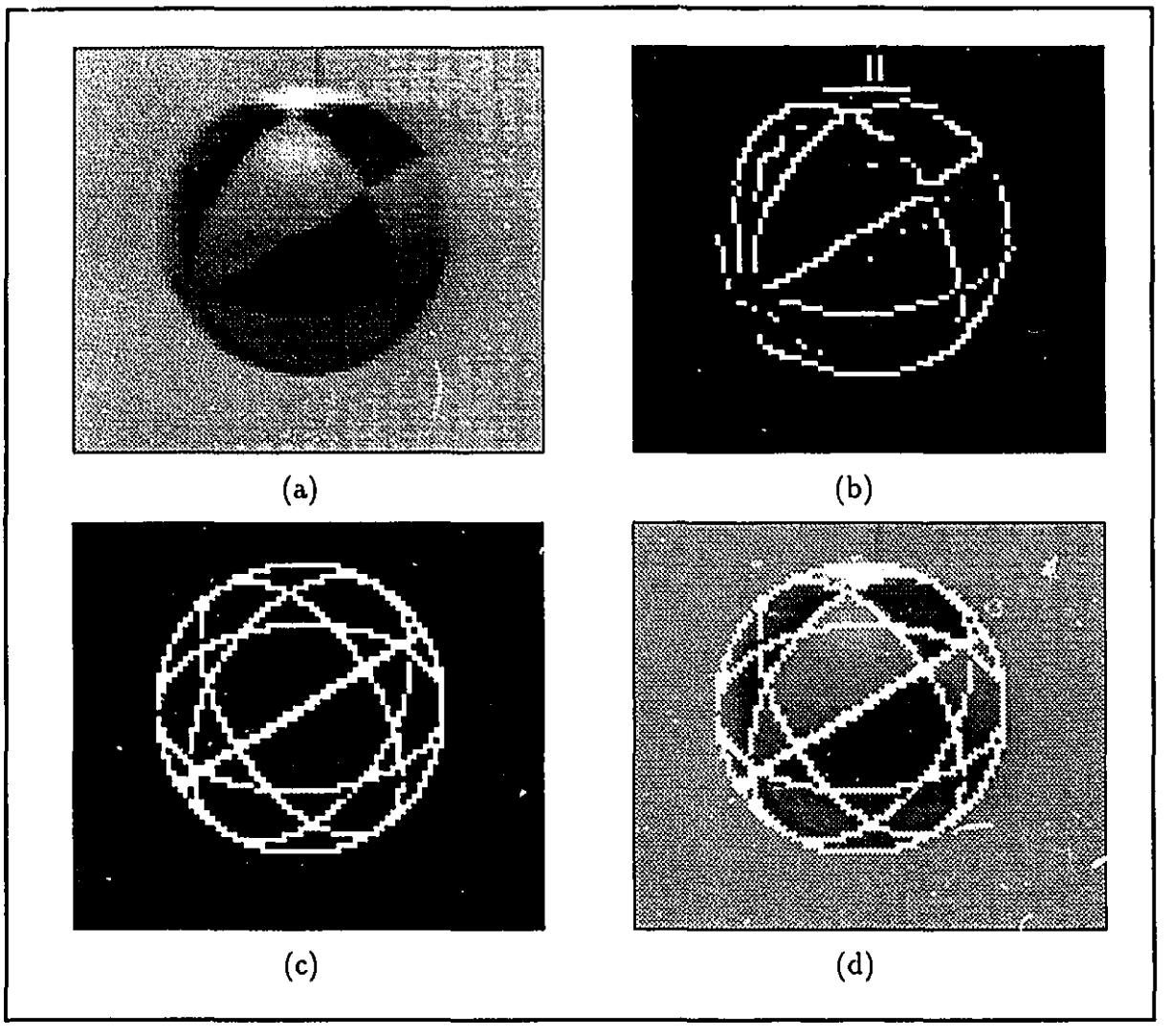

Figure 4.4: Extracting ellipses and circles (a) initial image (b) initial edge pixels (c) extracted ellipses and circles (d) extracted ellipses and circles superimposed on edge data.

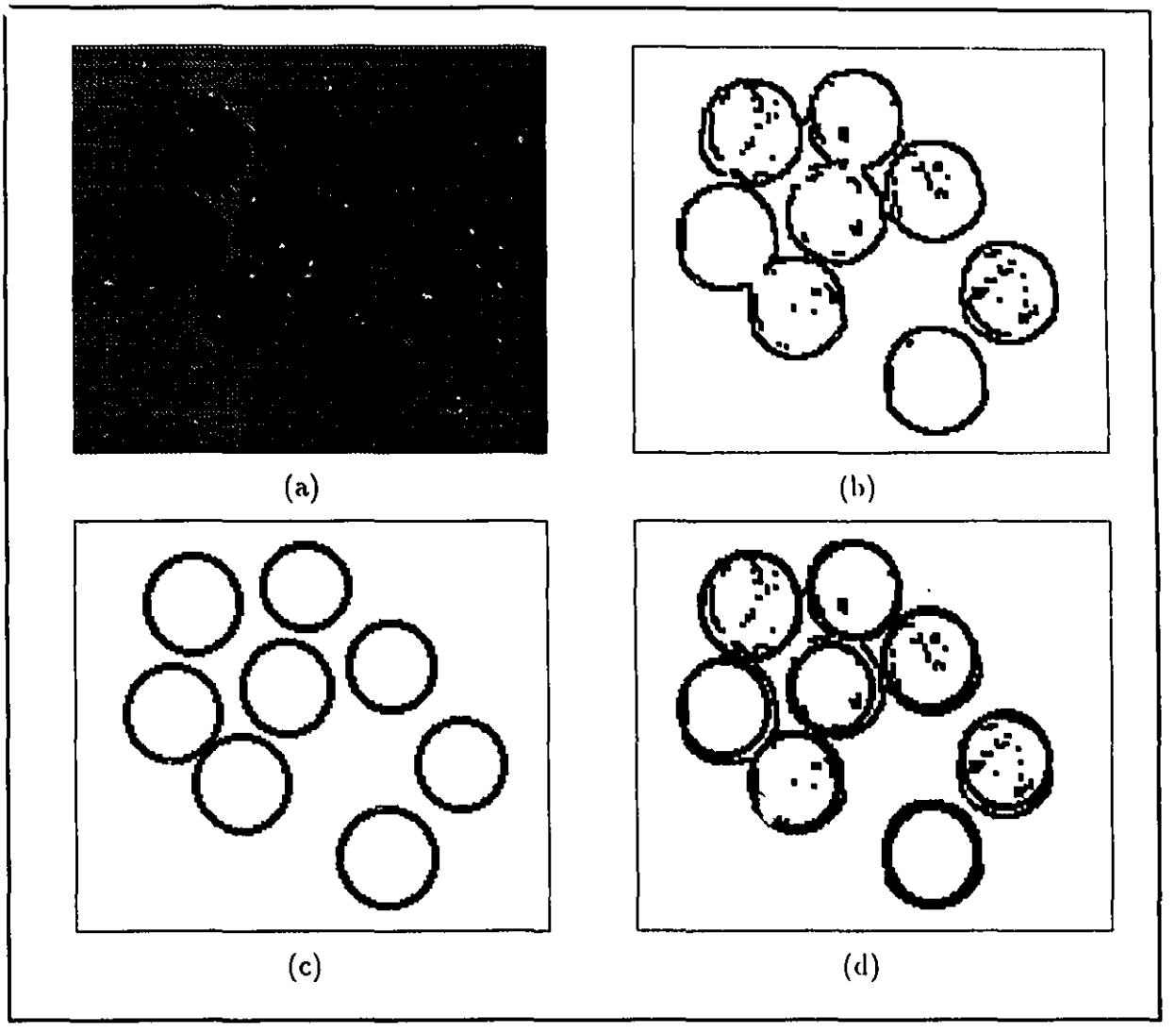

Figure 4.5: Extracting circles (a) initial image (b) initial edge pixels (c) extracted circles (d) extracted circles superimposed on edge data.

figure the correct circles have been found.

4.2.2 Three-Dimensional Data

Now we will discuss extraction examples from data taken from a variety of laser rangefinders. The first example has as input three-dimensional geometric data produced by a laser rangefinder mounted on a robot wrist [Rioux and Blais, 1986]. This rangefinder collects parallel profiles, where the number of profiles and the spacing between the profiles is controllable. Such data are relatively sparse since the spacing

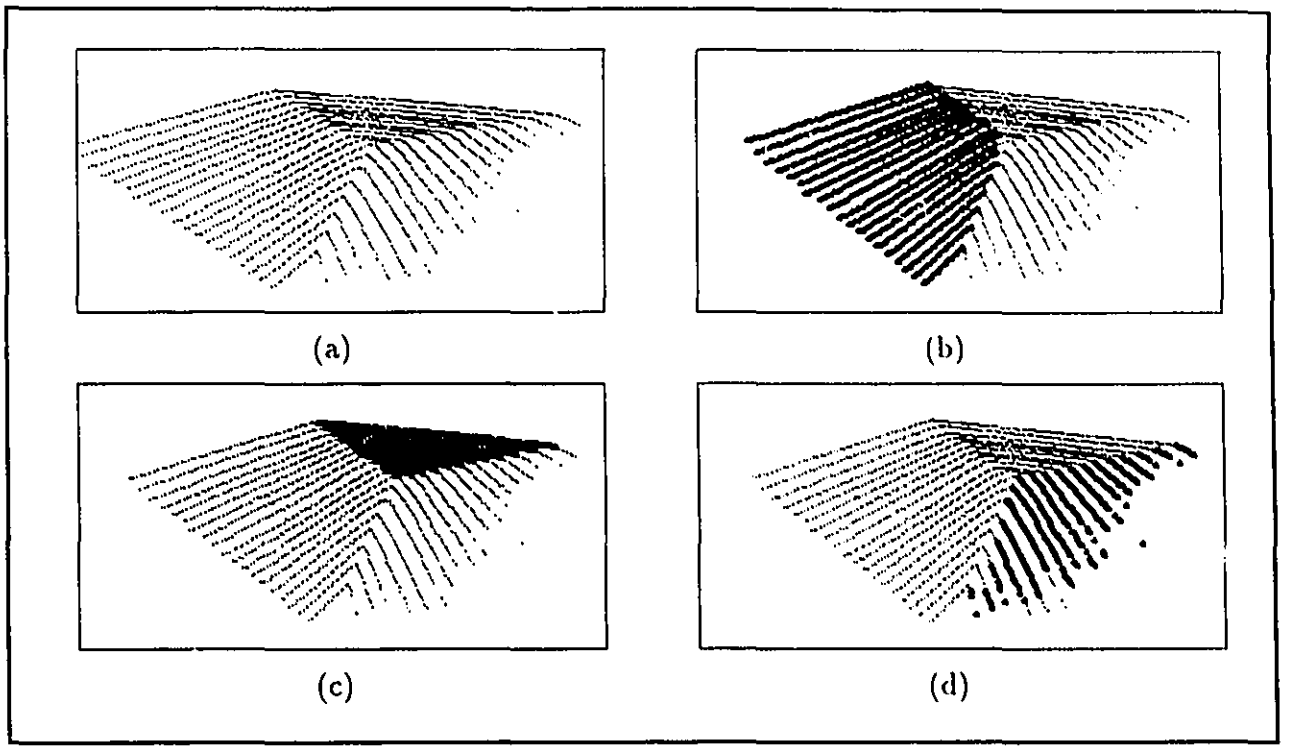

Figure 4.6: Plane extraction from range data (a) initial range data (b) (c) (d) extracted planes.

between profiles is fairly large. This example demonstrates the extraction of planes from such data. Three points defines a unique plane so a minimal subset is of size three in this example. The results are shown in Figure 4.6. Part (a) of the figure shows the initial range profiles, and parts (b), (c), and (d) show the points belonging to three extracted planes. These planes were extracted by repeated application of the minimal subset random sampling algorithm to all of the geometric data. Since the data is relatively sparse no connectivity information is available, nor is it necessary since the expected number of planes is small.

It is also possible to extract three-dimensional curves from these parallel profiles. In order to do this, a subset of the profile data, called the jump steps are used as the input geometric data points. Jump steps are points which have a significant difference in depth from their neighbours. These jump points define the three-dimensional outline of the object as seen from the rangefinder. In Fig. 4.7 a picture of an object called an "H-Fixture", which is the approximate shape and size of one of the grapple

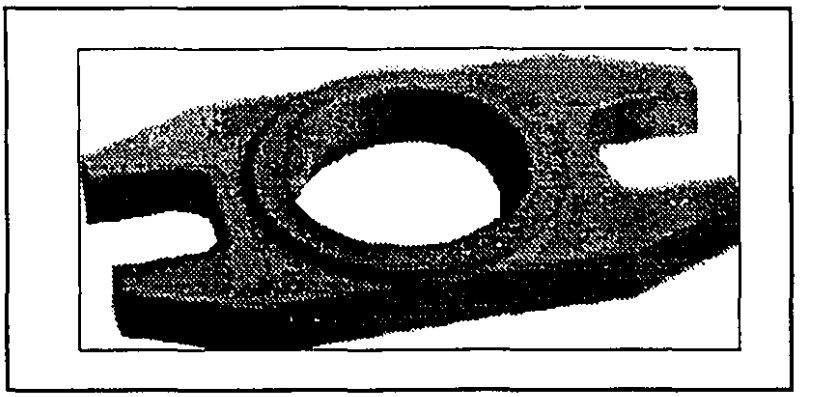

Figure 4.7: Picture of the H-fixture used for space grappling applications.

fixtures for the space station. In Fig. 4.8(a) are shown a number of parallel profiles of this object collected by a laser rangefinder. Fig. 4.8(b) shows the jump points obtained from this data, and these are points are highlighted as small dark circles. In Fig. 4.8(c) the jump points belonging to a three-dimensional line are highlighted, and in part(d) the jump points belong to a three-dimensional circle are similarly shown. In both cases these primitives were extracted using minimal subset random sampling, but with the jump steps shown in part (b) as input, instead of all the geometric data show in part (a). For a three-dimensional line the minimal subset is of size two, while for a three-dimensional circle the minimal subset is of size three. The equations relating the minimal subset to the parameter vector of the three-dimensional circle were derived in Chapter Three.

The next example shows the extraction of planar and quadric patches from dense range data. Such data differ from the previous examples because the collected profiles are now very close to each other. The data, as shown in Figure 4.9, consist of some planar blocks, a cylindrical block and a spherical block. This figure has four parts, with part (a) showing the original range image. Since the contrast of such images is low, they are redrawn in a shaded fashion in part (b). In this method of display the brightness is proportional to the angle the local surface normal creates with the viewing direction. This shading process makes it much easier to distinguish the contents of the images.

Since the amount of geometric data is very large, local connectivity is used to

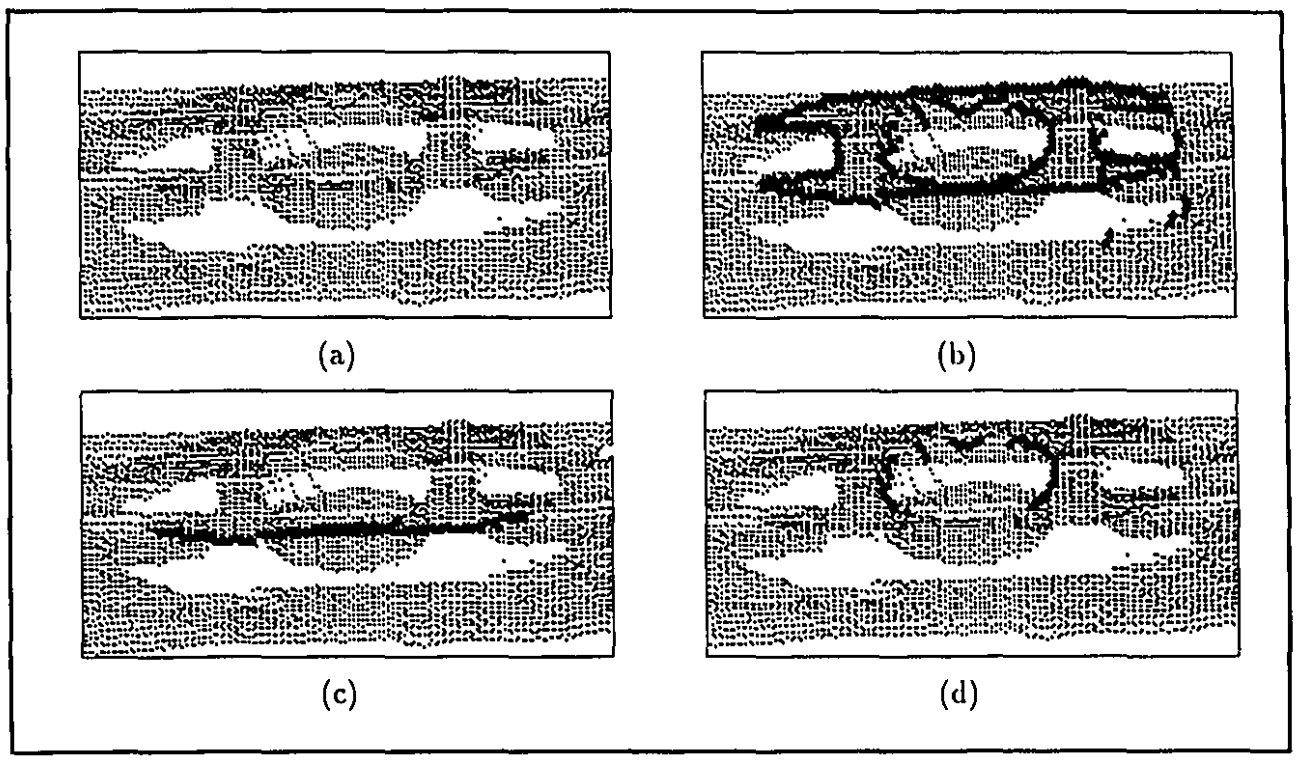

Figure 4.8: The points on a three-dimensional line and circle extracted from the jump step points (a) Initial parallel profiles of H-Fixture (b) Initial jump points (c) Three-dimensional line jump points (d) Three-dimensional circle jump points

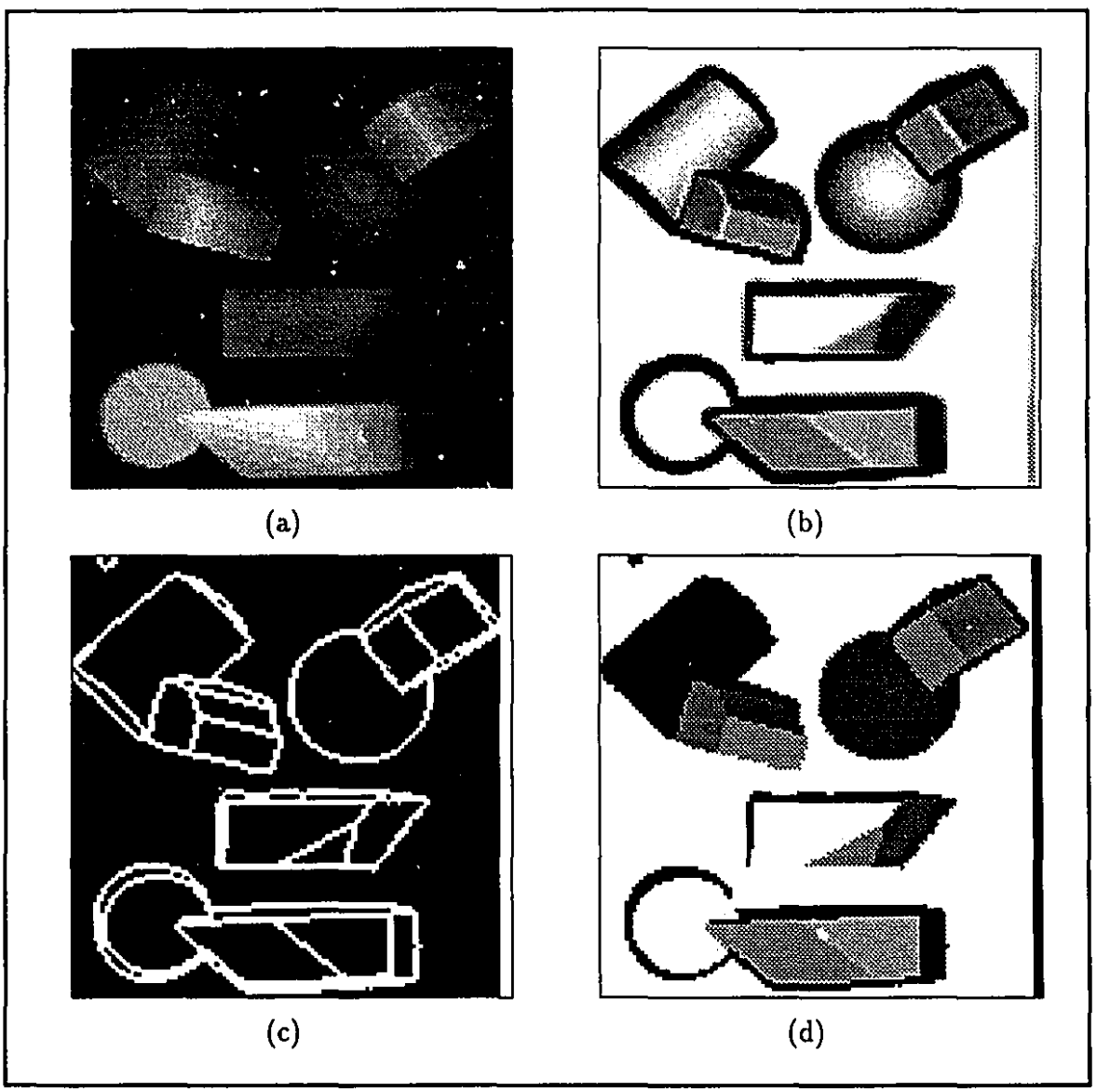

Figure 4.9: Plane and quadric extraction from range data (a) range image (b) shaded rendering of range image (c) region boundary pixels (d) shaded rendering of extracted planes and quadrics.

select subsets of the data to be processed by the extraction algorithm. This is done by finding the largest connected set of points bounded by jump and roof edge points in the range image [Godin and Levine, 1989]. The jump edge points are local discontinuities in depth while the roof edges points are local discontinuities in the normal. The method used to detect these edge points is very simple. The points marked as jump edges are those that have a difference in depth from their neighbours greater than a threshold, and the roof edge points are those that have a normal whose angular difference from their neighbours is greater than a threshold. The largest connected set of points not containing either a jump or a roof edge point are presented to the extraction algorithm. The extraction algorithm finds the planes and quadrics in this data, and the points belonging to the extracted primitive are removed. The process is repeated with the next largest connected set, and continues until no more points remain. The regions produced by the final segmentation are shown in part (d), while the boundary points between these extracted regions are shown in part (c). It can be seen that the segmentation is correct in terms of identifying the appropriate patches for each object. It should be noted that the black areas around some of the objects in part (b) are due to shadows [Godin and Levine, 1989]. These are artifacts of the range sensor which occur when the laser detector is blocked from seeing the projected laser spot by an object, i.e., the object occludes the laser beam. In the image under consideration, these shadow regions are interpolated in a nonstandard way, and they are not fit successfully by a plane or a quadric. For this reason, in Figures 4.9 (d) these regions are drawn in black to distinguish them from the other regions which have been successfully extracted.

Figure 4.10 shows the tolerance of the method to the setting of the threshold that computes the edge points which delineate the connected regions. The planar patches in the middle block of the image shown in Figure 4.9 are correctly found even when three different connected regions are produced by different edge thresholds. Figures 4.10 (a),(c),(e) shows the edges that are created by each of the three different edge thresholds. The largest connected region differs considerably for each of these thresholds. In Figure 4.10 (a) the largest connected region contains three faces, in Figure

4.10 (c) it contains two faces (there is a gap in the edge pixels), and in Figure 4.10 (e) it contains only one face. Figures 4.10 (b),(d),(f) show the pixels at the boundaries of the planar regions extracted by the segmentation algorithm using the associated edge threshold on the left. Even though only the last edge threshold (Figure 4.10 (e)) correctly isolates the three planar faces of the block, the final segmentation is the same for all three thresholds. Because of the inherent robustness of the extraction process, having different connected regions as input has not affected the final results.

4.3 Comparison with Other Approaches

In this section we will compare our extraction algorithms to other approaches. We will concentrate on comparisons with robust fitting, the minimal length encoding schemes and the Hough transform. These three methods are what we consider to be the closest competitors to our approach. Because of the ubiquitous nature of the HT, we will spend the most effort on a comparison with this approach.

4.3.1 Robust Fitting

The robust fitting community has concentrated on investigating the statistical properties of different cost functions [Hampel *et al.*, 1986, Huber, 1981]. In terms of the actual algorithms to compute the global optimum of these cost functions the suggestion is to run a non-linear optimization algorithm, in particular, the reweighted least-squares algorithm [Beaton and Tukey, 1974]. This algorithm, however, requires a starting point in the parameter space which is close to the global optimum in order for it to converge properly. But as we have shown, there are often many local optima, so finding the right starting point is not a simple task. This is where the minimal subset random sampling algorithm can be used effectively. The parameter vector of the best primitive produced by the random sampling process can be used to provide the initial estimate for the reweighted least-squares process. The quality of the results obtained by robust fitting algorithms that use reweighted least-squares depends on this initial estimate [Press and Flannery, 1988]. Therefore using random sampling to

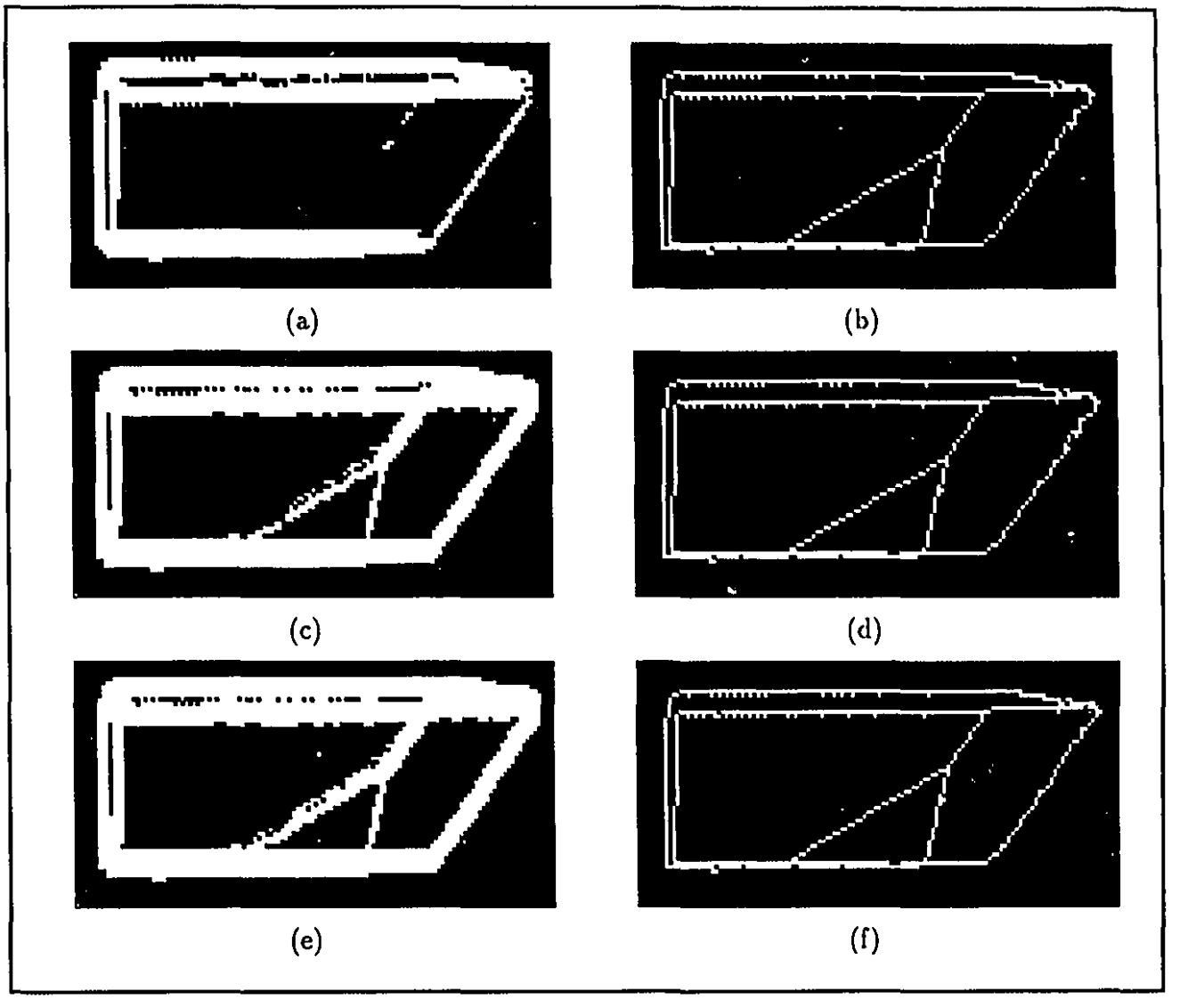

Figure 4.10: Block from range image (a),(c),(e) edge pixels from local operator (b),(d),(f) edge pixels from segmentation.

find an initial estimate for the reweighted least-squares process will definitely improve the chances of a successful result using any robust statistics fitting algorithm.

From a computational point of view, minimal subset random sampling is completely compatible with previous robust fitting algorithms. The real reason that these algorithms are not used for primitive extraction where there are more than 50% outliers is the concept of breakdown. We have shown that this idea is not completely transferable from statistics to the computer vision field. It ignores the fact that for any robust algorithm some threshold must be placed on the returned cost function value in order to decide on the significance of the result. We have discussed this issue at length in the Chapter Two. It is in the interpretation of breakdown that we differ from the robust statistics community. With our interpretation the essential unity of various extraction algorithms is clear. With the robust statistics interpretation the robust fitting algorithms and the extraction algorithms such as the HT are of a fundamentally different nature [Forstner, 1989]. From a statistical point of view this is true, but not from a computational point of view. However, even without agreement on this issue, minimal subset random sampling is still a useful numerical method that can be used with robust fitting algorithms.

4.3.2 Minimal Length Encoding

Recently approaches based on minimal description length (MDL) have become popular for segmentation [Darrell *et al.*, 1990, Pednault, 1989, Leclerc, 1989]. The MDL principle states that the best description is the one that is the shortest in a given language. In order to use the MDL principle it is first necessary to define a language along with a way of measuring the length of a string in that language. Once this is done, segmentation becomes an optimization problem where the attempt is made to minimize the sum two terms, $L(S) + L(E)$. Here $L(S)$ is the length of strings that define the segmentation, and $L(E)$ is the length of the segmentation error. The length in both cases is usually measured by computing the number of bits necessary to encode the given quantity. This makes the optimum segmentation the one that has the minimum length, where length is the total number of bits. This is the explanation of

the origin of the term "minimal length encoding". This technique has been applied to both two-and three-dimensional geometric data, though the three-dimensional data are assumed to be graph functions taken from a single view [Darrell *et al.*, 1990]. The main difficulty with the approach is the time complexity. Even simple segmentations of two-dimensional curves take considerable computer time.

Our way of applying the extraction algorithm to the problem of extracting different types of geometric primitives uses ideas similar to the minimal length encoding algorithms. At each stage of the extraction step we attempt to find all the geometric primitives. If two primitives are equally good according to the cost function value, then the simplest description is chosen. For example, if a set of points is planar, then both a planar patch or a quadric patch are equally good descriptions, but the planar description is simpler. Our measure of length (simplicity) is the size of the minimal subset. The fewer the points in a minimal subset, the simpler the primitive. Thus our segmentation algorithm does return the simplest description in a certain sense. However, it is not the best description according to the optimization criteria used in the MDL literature.

The MDL approach does not return a single primitive at a time, but instead returns all the primitives at once. This makes the associated optimization problem much more difficult to solve than for primitive extraction. The result is that MDL algorithms execute very slowly, even on massively parallel hardware. This is not acceptable for robot vision. The other major difficulty is that the MDL approach is only a general principle which leaves open exactly how the complexity and error are to be measured. While progress has been made in applying MDL to range images [Darrell *et al.*, 1990], it is too early to say that such an approach meets the segmentation criteria described in the previous section.

4.3.3 Hough Transform

The HT is historically the most common method of performing primitive extraction. It was discussed in detail in Chapter Two, where it was shown to be nothing more than a repeated template match [Stockman and Agrawala, 1977]. This means that

the HT is equivalent to a repeated application of primitive extraction with the appropriate fixed-band cost function. Because the HT is so common we will spend effort comparing it to our random sampling method. We will make this comparison in the categories of robustness, computational complexity, sensitivity to parameters, parallelism, and generality. We believe that for primitives other than lines our approach is definitely superior to the HT. For lines the issue is less clear, but we believe the random sampling algorithm is still preferable in many cases.

The first category of comparison is robustness, which according to our optimization model described in Chapter Two, depends on how well the global optimum can found from among many possible local optima. The HT attempts to find the global optimum by performing template matching (evaluating the cost function) only at cell locations. The likelihood that the good global optimum will be found depends on the cell quantization. Finer quantization of the Hough space mean a better chance of finding the global optimum, but requires more space than coarse quantization. Assume that the HT has a given cell size and that the band size for random sampling is set to produce the same size template. If the number of samples K of the random sampling approach is set equal to the number of cells of the HT, then random sampling is at least as likely as the HT (and probably more likely) to find the global optimum. This is because for random sampling the matching is done only at the values of the parameter vector defined by a minimal subset. It is at least as likely that a primitive will exist at the value of the parameter vector defined by a minimal subset than at an arbitrary location in parameter space defined by the cells of the HT. Therefore, for the same number of cost function evaluations our approach is at least as robust, and often more robust than the HT.

The next category is computational complexity, which are the time and space complexity of each algorithm. The space requirements of the HT are exponential in R, the degrees of freedom of the primitive, and the dimension of the parameter space. This is the most significant drawback of the HT. If each dimension of the parameter space is quantized equally into P values, then the space requirements are $O(P^R)$. By contrast, the random sampling approach requires only $O(N)$ space, where N is the

number of data points. This makes it possible to process primitives which have many more degrees of freedom than the HT.

This is most clearly shown in the application of the HT to circles and ellipses extraction, which is still an active area of research [Huang, 1989, Yuen *et al.*, 1989, Davies, 1987, Kierkeggaard, 1992, Yip *et al.*, 1992]. Because the number of degrees of freedom of these primitives is higher than two a direct implementation of the HT is not practical because of the space requirements. There are three approaches that have been used to modify the HT in order to deal with this problem. The first is to use a very coarse quantization of the parameter space. Then the HT for ellipse extraction have difficulty in finding closely separated ellipses because of the coarse quantization [Yuen *et al.*, 1989]. Our algorithm has no such difficulty and is able to incorporate available constraints on the geometric primitive in a natural fashion. The second approach is to make extraction using the HT a multistep process, where each step operates on some subset of the parameter space. However, this is clearly less accurate since parameters of a previous step do not necessarily provide the correct information for successive steps. The third, and most practical approach, is to use the gradient information at each edge point along with the edge point itself. However, since this gradient information is a derivative the result is less accurate than if just the edge points were use. Our extraction algorithm, by contrast, does not require gradient information, and is as robust as a higher dimensional HT.

The time requirement of both methods require more analysis in order to make a sensible comparison. For each data point, the HT increments the count of every cell in parameter space that could have produced this data point. Therefore, the time complexity depends on how many cells are visited for each data point. An exact answer depends on the primitive and its parameterization. However, an approximate answer comes from the realization that each data point is associated with a surface in the parameter space of the HT. If this space has dimension R then the largest subspace has dimension $R-1$. For example, in two dimensions, the number of cells is P^2 so the expected number of cells in any one dimensional surface (a curve) is P. Here P is the quantization of each dimension of the Hough parameter space.

Thus the expected number of cells visited by each point during the voting process is P^{R-1}. Since there are N points this makes the expected time complexity of the vote accumulation part of the HT equal to $O(NP^{R-1})$.

Once this voting process is complete it is still necessary to search the parameter space to find all the cells which are deemed to be significant. Traditionally, this process is called peak detection since the largest value, or peak, in the parameter space was is the most significant cell. In practice, the usual approach is that cells which have more than votes than a given threshold are returned as valid primitives. This process takes time proportional to the number of cells, which is $O(P^R)$. However, it is independent of N, the number of geometric data points.

Since the HT produces the same result as repeated template matching it is interesting to compare the complexity of a direct template matching approach to the HT. There are P^R possible templates (one for each cell) that can be matched against the geometric data. If the number of data points is N, then the time complexity for template matching is $O(NP^R)$, while the space complexity is $O(N)$. For the HT the expected time complexity is equal to $O(NP^{R-1})$ and the space complexity is $O(P^R)$. The extraction time of the HT is therefore proportional to N, the number of geometric data points. This shows that the HT is nothing more than a way of performing template matching that trades off time for space.

For random sampling with a fixed-band cost function each value of the parameter vector computed from the R minimal subset points defines a template. A direct implementation of the evaluation procedure for a fixed-band cost function requires $O(N)$ time. In Chapter Two we showed this can be improved upon, but for comparison purposes with the HT we will accept the more conservative $O(N)$ figure. In the worst case the number of random samples is $\binom{N}{P}$, and if we take N^P as an upper bound for $\binom{N}{P}$, then the worst case time complexity is $O(N^{P+1})$. This is equivalent to a direct template matching implementation of the HT with the quantization P set to the number of samples N.

However, we note that these figures are for the worst case, with the maximum number of possible random samples. Typically the number of random samples K is

considerably less since maximum robustness is usually not required. Thus the time complexity of the random sampling method is $O(KN)$. By comparison the expected time complexity of the HT is equal to $O(NI^{R-1})$. However, the HT finds all the geometric primitives at once, while our approach finds them one at a time.

It is also the case that the HT is most commonly used for the extraction of lines, circles and ellipses. For extracting these curves the method described in Section 2.4.3, of performing the template matching by incremental curve generation changes the execution time of our method to $O(KZ)$, where Z is a fixed value that does not depend on N. This means that as the number of geometric data points increases, the execution time of our extraction algorithm is unchanged. It is proportional to K, the number of random samples.

We see that a time comparison with the HT is difficult without specifying the expected number of primitives in the geometric data. The smaller the number of expected primitives, the smaller the value of K, and the faster our approach will be verses the HT. At some point as the number of primitives increases the HT will be faster than our approach. The number of primitives at which this occurs depends on the size of the minimal subset, but in general is in the order of a dozen primitives. However, for 2D curve extraction if the number of primitives remains fixed but the number of geometric data points increases, then at some point our method will be faster than the HT. This is because the execution time for the HT is proportional to the number of geometric data points. When incremental curve generation is used for 2D curve extraction this is not the case.

The previous analysis assumed that both the HT and the random sampling method were running on a single processor with no special hardware. However, if real-time performance is to be obtained special purpose hardware will often be necessary. In the previous Chapter we showed that using VLSI implementations of incremental curve generation routines would enable real-time extraction of lines,circles and ellipses. This can be achieved without parallel hardware, which is not the case for the HT. Since extracting such curves is the most common application of the HT, this is an important fact. However, the incremental drawing approach applies only to

2D curves, for other types of geometric primitives we require parallel hardware to achieve real-time performance. The issue of running the random sampling algorithm on parallel hardware will be dealt with in detail in Chapter Six. Here, we will provide a short summary of our conclusions from this Chapter.

For the HT, the obvious approach would be to dedicate a processor to each point. If the number of processors equals the number of points then the maximum possible speedup is obtained. However, because of memory contention the time to access global memory in a parallel architecture increases with the number of processors. Since the cells of the HT must be kept in global memory, each processor must be able to access this memory to perform the voting process. This rapidly leads to contention, and decreases the potential speedup which can be realized for multiple processors. Any algorithm which has a large number of accesses to global memory is very difficult to parallelize. In fact, parallel approaches to the HT have only been implemented on special purpose hardware for line extraction [Fisher and Highnam, 1989]. This implementation exploits the fact that for lines both the image array and the Hough array are two-dimensional. This scheme does not seem to be extendable to higher order primitives. The fact that the HT uses a large amount of global data makes it very difficult to parallelize [Illingworth and Kittler, 1988].

The random sampling approach can be parallelized most directly by concentrating on the ranking step. Normally this takes $O(N)$ time, but a direct implementation using N processors can decrease this time to $O(1)$. This assumes that a global broadcast facility exists in which a message can be sent in fixed time to all the processors. Significant speedup is still possible if such a facility does not exist. Because the random sampling algorithm requires no access to global memory it is easy to decrease its execution time on parallel hardware. With the requirements for global memory this can not be said of the HT.

The next criterion for comparison is the number of parameters on which each algorithm depends and the sensitivity to the settings of each parameter. As we have stated earlier, the key parameters in the HT are the parameterization of the geometric primitive, the cell size and the cell shape. The cell size is a choice of scale and as

such must be set by the user, but the others are arbitrary and have a dramatic effect on the results.

The random sampling approach has as its operating parameters the number of samples, and the band size. The latter is equivalent to the cell size of the HT. However, unlike the HT the random sampling method does not depend on the parameterization of the primitive. For example, whether a line is parameterized in slope-intercept form or in distance-angle form has no effect. As opposed to the HT, the template always has the correct shape, the shape of the desired primitive.

The kinds of primitive that can be extracted by each method is what we mean by the generality. The HT votes for each point by traversing parameter space in a way defined by the original implicit form of the geometric primitive. The difficulty is that this traversal process is not always simple. It is more appropriately done using parametric equations, but producing a parametric form from the implicit form is not always possible [Sederberg and Anderson, 1984, Hoffman, 1989]. The traversal process is possible if the implicit form is a first or second order polynomial, since in this case the associated parametric form can easily be produced. Because only a very restricted subset of all possible surfaces has been processed with the HT, the difficulty of actually voting for each point by traversing parameter space has not been a problem, but it is in fact a limitation of the method.

One suggested way of making the HT more general is to use a table containing a list of points on the outline of an object along with their distance and angle from a reference point. The peaks in the accumulator are then the possible locations of the reference point. This is known as the generalized HT [Ballard, 1981]. However, this method is applicable only to a single instance of a given shape, and is not capable of extracting an entire class of shapes defined by the implicit form. For example, this method could extract an ellipse of a particular size and orientation, but would not be effective for any other ellipse. The generalized HT can not extract different families of curves or surfaces, where a family is defined implicitly by a given equation. Thus, it does not actually deal with the problem of extraction, which is the focus of our algorithm.

The random sampling algorithm requires the ability to produce the parameter vector of the primitive through R points. The details of our approach for converting from a minimal subset to a parameter vector were discussed in Chapter Three. In that chapter we discussed the class of primitives for which we can currently perform this mapping, and this class is large enough to contain a wide variety of geometric primitives.

We believe that our algorithm is simpler and more general than the HT. It can handle a wider variety of geometric primitives. Its main drawback is the execution time, while the main drawback of the HT are the space requirements. Thus as processor speed increases our method will become more competitive. This is not true for the HT, since for complex primitives its space requirements are difficult to meet even on future processors. For a reasonably small number of primitives, often in the order of a dozen, the execution time of our method is comparable to the HT. In many situations both will require implementation on parallel hardware, and here our method has the advantage over the HT. This issue is so important that it is dealt with in detail in Chapter Six, which discusses parallel hardware implementations of our approach.

4.4 Summary

The extraction algorithm described in the previous chapter finds a single geometric primitive. In this chapter we showed how to apply it when there are multiple primitives of the same type or of different types. The most straightforward approach is to repeatedly apply the extraction algorithm and remove the inliers found at each iteration. This process continues until no more primitives are found. Sometimes the number of primitives in the geometric data is so large that very many random samples are necessary. In this case we use local connectivity to divide the geometric data into components, and then perform extraction on each connected component. A number of experimental results for both two-and three-dimensional geometric data are shown. They show that extraction can be performed successfully on a wide variety of

curves and surfaces. Finally a detailed comparison is made between our approach and the robust fitting methods, the Hough transform and the minimal length encoding methods.

Chapter 5 Genetic Algorithm for Extraction

In this chapter we will discuss the use of a Genetic Algorithm (GA) for primitive extraction. Since GAs have rarely been applied in the computer vision field we will first give a general overview of their operation. Then we will show how a GA can be combined effectively with the minimal subset representation [Roth and Levine, 1991a]. The resulting approach is often much more efficient than the extraction algorithm that uses only random sampling. The GA combines inaccurate local information to create a more accurate global interpretation in a natural fashion. The GA does this by emulating the process of evolution to improve the quality of a set of potential solutions. While convergence to the best solution is not guaranteed, a very good solution is often quickly found.

The combination process is reminiscent of a traditional computer vision grouping algorithm [Levine, 1985]. However, such algorithms generally have a complex control structure, and also suffer from the problem of premature commitment. A GA is a form of directed random search or learning, and this random component of the search process means that in many cases premature convergence is avoided. A GA also has a very simple control structure in which the best geometric primitive emerges spontaneously. We spend some time discussing why the GA works, and compare it to other search methods such as simulated annealing. Some experimental results are presented for primitive extraction using a GA, and the parameters settings of the GA algorithm are discussed in detail.

5.1 Genetic Algorithm Overview

A GA is a method of solving hard optimization problems in a way that attempts to mimic nature. It is based on an evolutionary metaphor, which has its own terminology, so some explanations of this terminology are necessary. In evolution, a population of individuals selectively mates, and in doing so evolves. This evolutionary process

is guided by feedback from the environment. This means that fitter individuals live longer, and therefore reproduce more often. As time goes on the population becomes dominated by these fitter individuals and their descendents. Conceptually, a GA is nothing more than a procedural codification of this evolutionary process, albeit in a simplified form [Holland, 1975].

A GA has the following three essential components [Goldberg, 1988]:

- A population of individuals which are candidate solutions to the given optimization problem.
- A competitive selection method for choosing individuals for reproduction, based on the fitness of each individual.
- A set of genetic operators that combine the selected individuals to create new individuals for further testing.

Individuals are defined by a chromosome string which is a list of genes, where each gene takes on a single value from a set of tokens. The chromosome encodes all the possible values of the parameter(s) being optimized. In traditional GA literature the set of tokens for a gene is either zero or one, which makes the chromosome a bit string. Instead of using a binary encoding we use the minimal subset encoding discussed in previous chapters to define a chromosome. Since the basic operation of the GA does not depend on the particular chromosome representation, we will postpone discussion of the representation issue until later in this chapter.

Feedback from the environment is provided by means of a fitness function. This function has as input the chromosome definition, and as output a scalar whose magnitude is the fitness of the particular individual. The fitter the individual, the larger this scalar value. How this fitness function is implemented is completely application-dependent. In optimization terms this fitness function is nothing more than the cost function. Thus the GA approach is naturally adopted to optimization problems. In fact, the optimization field is the one in which the most experience with GAs has been obtained [Goldberg, 1988].

In order to explain the actual operation of the GA we will assume that a population of individuals exists, along with their fitness values. Later we will show how this *initial* population of individuals is obtained for our application. During execution of the GA two individuals are repeatedly selected at random from the current population. The key point is that this random selection process is not uniform, but is in proportion to the fitness values of the individual. Let us say that $f(M)$ is the fitness of a given population member, and $f(T)$ is the sum of the fitness values for all the population members. Then this population member will be selected with a probability $f(M)/f(T)$. Thus the probability of selection is "biased" by the fitness value of that population member. This means that fitter population members are selected to mate more often, which is how evolution operates in the natural domain.

The GA takes two individuals selected in this fashion and applies genetic operators to their chromosomes to create two new individuals. The most important of these operators is called crossover. This takes the chromosomes of two individuals (the parents) and crosses them over to create the chromosomes of two new individuals (the children). For example, assume that the parents have binary chromosomes of (010110) and (110011). Then a single application of crossover might produce two children, whose chromosomes are (110010) and (010111). In this example the genes in the first and third position of the parents' chromosomes are crossed-over or switched. The actual genes which are crossed-over are also selected randomly according to an algorithm which we will describe. Figure 5.1 illustrates how crossover is applied to the parents' chromosomes. Here the chromosomes of two parents are drawn on top of each other, one shaded in solid black, and the other shaded in a cross-hatched pattern. The crossover operator crosses-over the parents' chromosomes in a random fashion to create the chromosomes of two children. This figure shows one possible set of children's chromosomes produced by the crossover operation.

The second genetic operator is called mutation, and is nothing more than a random change of the genes in a chromosome. For example, a chromosome string of (110110) might become (010111) by the mutation of the first and last gene. The mutation operator is applied to the children's chromosomes after crossover. There are

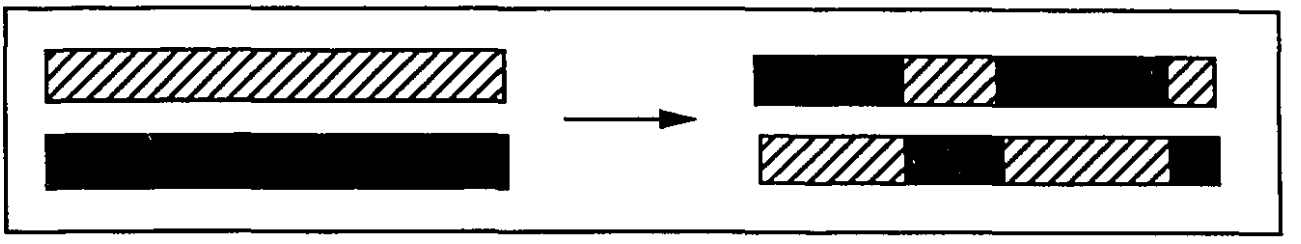

Figure 5.1: Crossover operating on the parents' chromosomes to create the children's chromosomes

other more specialized genetic operators, but they are not always present in any GA implementation. The operators of crossover and mutation constitute the core genetic operators. Of the two, crossover is the more important, so much so that some GAs perform no mutation at all. In our application this is indeed the case, and we do not use the mutation operator. Mutation is useful only for an application in which the GA is run for many iterations. This is not the case for primitive extraction, since the number of iterations of the GA is relatively small.

The fitness of the two children is evaluated using the cost function, and they are returned to the population pool. In order to keep the population at a fixed size, the two least fit individuals are removed. The GA allows identical population members, and in fact population members generally become more similar as time goes on. This is because the population becomes dominated by a small number of fit individuals. The GA loop of selection, crossover and mutation is executed until the fittest population member has not changed in a given number of iterations. When this occurs the GA is said to have converged.

The following pseudo-code describes the basic GA loop.

Create the initial population members
For a number of iterations
Do
1. *Choose two parents to mate in proportion to their cost function value.*
2. *Apply genetic operators to the parents' chromosomes to create two children.*
3. *Rank the children by using the cost function and add them to the population.*

Enddo

When the algorithm is terminated the fittest population member is taken as the solution. The GA acts as a form of adaptive search, but one in which the adaptation

proceeds in an evolutionary fashion. This basic methodology, with some variations has been applied to a wide range of problems, often quite successfully [Goldberg, 1988]. The question that needs to be answered is when and why this approach is superior to traditional optimization measures such as gradient descent, or other stochastic approaches such as simulated annealing [Corana *et al.*, 1987].

In relation to gradient descent methods the answer is clear. For cost functions which are noisy, and have many local minima, a gradient descent procedure will not be effective. In order to avoid local minima it is necessary to have a degree of randomness in the search procedure. Any deterministic approach to searching such a function is likely to be trapped in a local minimum. In computer vision terms, being trapped in a local minimum is an example of premature commitment to a particular solution. This phenomenon has been a problem with many computer vision algorithms in the past. The GA does not guarantee that this will not happen, however, in practice, random selection based on fitness often avoids premature commitment.

The answer to the second question is more subtle. There are other random search procedures, such as simulated annealing, which are also used in the optimization field. When and why is a genetic algorithm superior? There are both theoretical, and practical answers to this question.

5.2 Schemata and Their Implications

The main tool that has been used to achieve this understanding is the concept of schemata [Holland, 1975]. Schemata are simply generalized chromosomes. Normally the tokens for each gene in a chromosome are drawn from a given set. For schemata, an extra token, which is given the label $*$, is added to this set. For example, if the token set for a gene were $\{0, 1\}$, then the token set for schemata would be $\{0, 1, *\}$. The $*$ is a don't care, or wild card symbol that matches any of the other tokens in the set. An individual instance of a schemata is called a schema. It represents all the chromosomes that are matched by the schema pattern. For example, using binary tokens the schema $(1 * 0)$, matches the chromosomes 100 and 110. There are

many possible chromosomes matched by each schema and there are many possible schemata. If the chromosome has l genes, and there are normally k different tokens for a gene, then there are $(k+1)^l$ possible schema.

To illustrate this concept we will list all the possible schemata for a binary chromosome of length two. There are nine possible schemata, four of which are the chromosomes $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$, which are simply schemata without a $*$ token. The schemata that contain a $*$ are $(*,*)$, which matches $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$; $(1,*)$, which matches $(1,0)$ and $(1,1)$; $(0,*)$, which matches $(0,0)$ and $(0,1)$; $(*,0)$, which matches $(1,0)$ and $(0,0)$; $(*,1)$, which matches $(0,1)$ and $(1,1)$. Since schemata are generalized chromosomes, they can represent different regions of the search space. These regions can be very general (the entire space, which is a schema with all $*$'s) or very specific (a single chromosome, which is a schema with no $*$'s). Schemata will be used to show how random selection according to fitness, along with application of genetic operators, directs the GA search process. However, before these issues are explored, some further notation needs to be defined. When a single schema is referenced it is given the label H. The order of the schema H is called $o(H)$. This notation o, should not be confused with the computer science notation O, which represents algorithm complexity. $o(H)$ equals the number of fixed (non $*$) positions in the schema. For example, with the schema $(1*0)$, the order is two. The order is an integer from zero to l, where l is the length of the chromosome. Assume that the current population operated upon by the GA has r members, and that it is given the label A. Let $A(t)$ be the population of the GA algorithm at iteration t. The variable t starts at one, and increments by one for each iteration of the GA. Here an iteration is the process of selecting two parents and creating two children by applying genetic operators. The size of the population A stays constant, but the composition changes as t increments. Let $m(H,t)$ be the number of population members matched by schema H in the population $A(t)$. For example assume the population at iteration t is $(101),(001),(110)$, and the schema H is $(*01)$. Then $m(H,t)$ equals 2, since only population members (101) and (001) are matched by H. Clearly, $m(H,t)$ is always less than or equal to r, the number of members in the GA population.

Recall that an individual population member will be selected for mating with a probability of $f(M)/f(T)$ where $f(M)$ is the fitness of the population member, and $f(T)$ is the sum of the fitness values for all population members. Since the rule of selection according to fitness also applies to the population members matched by a schema, we expect the following equation holds:

$$m(H, t+1) = m(H, t)\frac{rf(\hat{H})}{f(T)} \tag{5.1}$$

In this notation $f(\hat{H})$ is the average fitness value of all the members of the population $A(t)$ matched by schema H, so $rf(\hat{H})$ is the total fitness of all the members matched by this schema. Since the average fitness of the entire population $f(\hat{T})$ is defined by $f(\hat{T}) = f(T)/r$, the previous equation becomes:

$$m(H, t+1) = m(H, t)\frac{f(\hat{H})}{f(\hat{T})} \tag{5.2}$$

This equation is the basic reproduction equation of a GA [Holland, 1975]. It says that the number of population members matched by schema H grows in proportion to the ratio of the fitness of the average schema member to the average fitness of the population. A Schema which has above average fitness will have more representatives in the population as time goes on, and those with below average fitness will have fewer. Assume that a particular schema always has a fitness value which remains above the average fitness by a constant amount c. Then $f(\hat{H})$ equals $f(\hat{T}) + c\,f(\hat{T})$ and the above equation becomes:

$$m(H, t+1) = m(H, t)\frac{f(\hat{T}) + c\,f(\hat{T})}{f(\hat{T})} = (1+c)\,m(H, t) \tag{5.3}$$

If we start at $t = 0$, and assume a constant value of c then equation (5.3) becomes:

$$m(H, t) = (1+c)^t m(H, 0) \tag{5.4}$$

This is simply a geometric progression over t, which is the discrete version of an expo-

nential equation. This shows that the selection by fitness allocates an exponentially increasing (or decreasing) number of population members to schemata that are above (or below) the average fitness value. Since a schema can represent regions of the search space, this implies that regions which have on the average very fit members are rapidly allocated more attention by the GA. This explains why such a simple reproduction scheme converges so quickly.

While interesting, this analysis has so far neglected the effects of genetic operators by assuming that at each GA iteration the children are simply duplicates of the parents. This is not true because genetic operators are applied to the parents' genes in order to create the children. After their application the children may no longer be members of the same schema as their parents. Thus the general reproduction equation must be multiplied by a factor s to become:

$$m(H, t+1) = m(H, t)\frac{f(\hat{H})}{f(\hat{T})}s \tag{5.5}$$

In this equation s is called the survival probability. This is the probability that after the genetic operators have been applied the new population members are still matched by the given schema H. This will happen only if the genes of non $*$ members of the schema are unchanged by these genetic operators. For example, a chromosome of (101) is matched by a schema of ($*$01). Assume that the crossover operator had this chromosome as one of the two parents, and that one of the resulting children had a chromosome of (111). The second gene of the chromosome changed, and since it is not matched by a $*$ in the schema definition, this child is no longer a member of the schema H. The value of s can be computed, and doing so gives considerable insight into the how genetic operators work.

Before this is done the operation of the crossover operator will be explained in more detail. As stated previously for primitive extraction the only genetic operator used is crossover, and no mutation is performed. The justification is that the number of iterations of the GA necessary in this application is too small for mutation to have a significant effect. A uniform crossover operator is used, which means that

there is a uniform probability p_c, of crossing over any of the parents' genes. In this application p_c is set to .5, so there is a 50% chance of crossing over any given gene. This is not the usual crossover operator, but a growing body of theoretical and practical work points to its superiority over traditional GA crossover operators [Syswerda, 1989, Spears and DeJong, 1991]. However, the basic conclusions of the schema analysis are the same even when more traditional genetic operators are used [Holland, 1975, Holland, 1992, Goldberg, 1988]. The final difference from a standard GA is that the crossover operator is always applied to the parents, instead of applying it only a certain percentage of the time. Again, the justification is that since there are a relatively small number of iterations of the GA, crossover should be done as often as possible. Assume that two parents are chosen randomly in proportion to their fitness. The following pseudocode shows how crossover is applied to the parents' chromosomes to create the children's chromosomes.

For Each Gene of the Two Parents' Chromosome
Do

1. *Draw a random number from zero to one.*
2. *If this number if less than p_c, the two children's genes are the same as the parents; otherwise they are the parents' genes crossed-over.*

Enddo

The result of this process are the chromosomes of the two new children.

Now the question is how this crossover operator affects the basic reproduction equation. Remember that $o(H)$, the order of schema H, is the number of non $*$ positions in the schema. To make a schema that matches the parent's chromosome to fail to match the child's, the crossover operator must change a gene which does not have a $*$ in the schema. The probability of this occurring, i.e., of all $o(H)$ non $*$ genes surviving crossover unchanged, is $(1-p_c)^{o(H)}$, where p_c is the probability of crossover changing a gene. This is the survival probability s and when this value is substituted in equation (5.5) it becomes the following:

$$m(H,t+1) = m(H,t)\frac{f(\hat{H})}{f(\hat{T})}(1-p_c)^{o(H)} \tag{5.6}$$

This new schema equation shows the effects of crossover on the number of population

members matched by a given schema, which is $m(H,t)$. Crossover tends to destroy (or disrupt) a long schema, where long means one that has a large values of $o(H)$. Disruption is a decrease in the number of individuals in the population that are matched by the schema. This means a disruptive process will make $m(H,t+1)$ less than $m(H,t)$. The longer the schema, the greater $o(H)$ and the greater the chance of disruption, and the less the chance of schema survival. The probability s of surviving crossover decreases exponentially as $o(H)$ increases. However, this disruptive tendency is counterbalanced by the basic reproductive process, in which above average schemata (those with large $f(\hat{H})/f(\hat{T})$) dominate the population at an exponentially increasing rate. What can be concluded from this? That short schema of above average fitness are the ones that will be favoured by a genetic algorithm. These represent regions of the search space where good solutions cluster, and these regions are searched more thoroughly by the GA. Thus the rate at which a GA samples different regions of the search space depends on the probability of finding a good solution in that region. The higher this probability, the more attention the GA expends in searching this region. This is exactly what one would like in a search procedure. In contrast, a completely random search process places equal effort in all parts of the search space regardless of how promising the solutions are in any given area.

Short schemata of above average fitness are the basic building blocks of a GA. Because they rapidly dominate the population, the crossover operator repeatedly combines them to try and produce fitter individuals. This analysis also suggests the type of problem for which a GA will not work well. Assume the best solution stands alone as a "spike" in the search space, and has neighbours of below average fitness. Then the fitness of the population members matched by a schema containing this solution and its neighbours will likely be below average. This means the GA will probably not find the best solution, since it will not spend much effort on the part of the search space represented by a below average schema.

It is clear that the success of a GA depends critically on the effectiveness of the crossover operation. The "building blocks hypothesis" says that crossover helps when

short, fit schemata combine to form even more highly fit schemata [Holland, 1975]. Whether this actually happens depends on the problem and the choice of chromosome representation. The choice of a GA representation appropriate for a particular application is still largely an experimental process. The claim is that for the minimal subset representation this "building block hypothesis" holds, and this will be demonstrated experimentally. If the "building blocks hypothesis" does not hold then the crossover operator accomplishes little, and a GA is likely to be no better than a purely random search procedure.

This leads back to the original question of how well a GA performs relative to other stochastic search procedures, in particular, in relation to simulated annealing. The expectation is that, for a good choice of chromosome representation, a GA will be superior, since the "building block hypothesis" will be true. However, the issue of whether a GA is superior to simulated annealing is far from being decided [Bramlette and Cusic, 1989]. What is accepted by both sides in the debate is that a GA is inherently a parallel process, while simulated annealing is inherently sequential. Thus a GA can use parallel hardware to advantage, and it is not clear whether this is the case for simulated annealing. This reason alone may be sufficient to prefer the GA.

5.3 GA Applied to Primitive Extraction

In this section we show how the GA is applied to the problem of primitive extraction. In Chapter Two we discussed how random sampling using minimal subsets can solve this problem. Here, we will describe the GA extraction algorithm in which minimal subsets are used as a chromosome representation. Traditionally, GAs have used bit strings to represent chromosomes where each gene is one of two tokens, either zero or one. We will show that for accurate geometric data the minimal subset representation is superior to the bit string representation for this application. First, we will describe the minimal subset chromosome representation in detail.

As defined in Chapter Two, a minimal subset is the smallest subset of geomet-

ric data that produces a unique geometric primitive. We assume that there are N geometric data points in the input. Each point is identified by its index, which is a number from 1 to N. Therefore the points in a minimal subset are defined by a list of their indices. For example, assume there are a total of ten input points and that points one, six and nine are the minimal subset points defining a circle. Then the minimal subset representation of this circle is $(1, 6, 9)$. This minimal subset representation is a chromosome if we think of each point as a gene. The length of the chromosome, that is the number of genes, is equal to the size of the minimal subset. The token set for each gene is then $\{1, \ldots, N\}$, instead of $\{0, 1\}$. Note that this chromosome definition is order independent, since any permutation of the genes produces the same geometric primitive.

This chromosome representation is not the obvious way to represent a geometric primitive, which is to encode the parameter vector $\overline{a}$ as a bit string. However, in this case every possible value of the parameter vector $\overline{a}$ would be a potential solution. When using a minimal subsets with P elements, only the $\binom{N}{P}$ values of $\overline{a}$ defined by each minimal subset are possible solutions. This is essentially the same reason that we use minimal subsets for primitive extraction in Chapter Two. The disadvantage of using minimal subsets is that as the accuracy of the data decreases, the chance of the best primitive being defined by a minimal subset also decreases. However, as we have argued in previous chapters, for modern sensors the accuracy of the geometric data is usually good. This means that the best geometric primitive is likely to be very close to the one defined by a minimal subset.

The other component necessary for a GA is a fitness function. This takes the chromosome defining the primitive, and outputs a scalar which is the fitness of that individual. For primitive extraction the fitness function is equal to the cost function used in our optimization model for extraction described in Chapter Two. While many cost functions for extraction were discussed in chapter two the simplest counts the number of data points in a fixed-band template around the curve or surface. This is a sensible way to score a geometric primitive, since the more points matched by it, the less likely that this alignment of points is random, and the better the chance

of a valid primitive. Since this cost function is the most common one we have used previously, we will also use it in our examples in the next section. However, any cost function defined in Chapter Two could have been used.

We will now describe each of the steps in the basic GA algorithm as applied to the problem of primitive extraction. The first step is the creation of the initial population for the GA. For the extraction application, we divide the geometric data into parts, where each part contains a relatively small number of points. To create the initial population of geometric primitives random sampling of minimal subsets is performed on each part. Since these samples are taken over a small amount of geometric data the points on a minimal subset have a good chance of falling on a single primitive. However, since the minimal subset points are close together the primitive defined by these points is not likely to be accurate.

There are different ways of partitioning the geometric data. If connectivity information is available, then standard procedures exist to find connected components. For example, if the geometric data consist of edge points from an intensity image created by a digitizer, then a simple algorithm produces connected chains [Giraudon, 1987]. Random sampling of the points on each chain can be used to create the initial population of geometric primitives. If such connectivity information is not available, then a hierarchic data structure, such as an octree or a k-d tree [Samet, 1984], can be used to partition the geometric data. In the same way as for chains, random sampling of the points on each terminal node of the data structure can be used to create the initial population of geometric primitives. In our experiments, we will demonstrate both approaches to creating the initial population for the GA. Other approaches could also be used; all that is important is that the initial population be chosen over a local portion of the geometric data. In this case the chance of a randomly sampled minimal subset falling on a single primitive is high, so a relatively small number of random samples is necessary to create the initial population [Roth and Levine, 1990b]. However, as we stated in the last paragraph, these locally sampled minimal subsets are unlikely to produce the best descriptions of a geometric primitive for anything except perfectly accurate data.

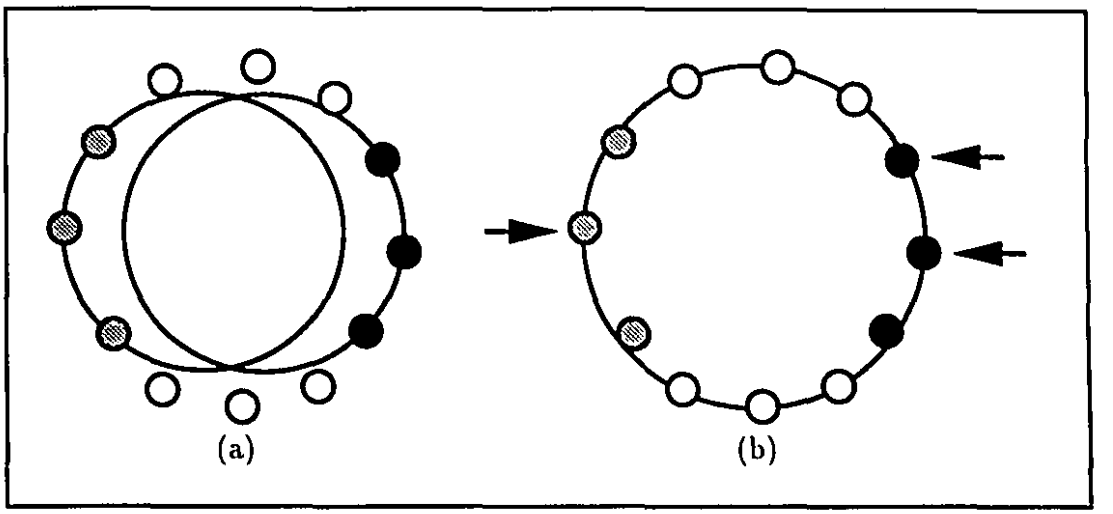

Figure 5.2: The crossover operation (a) Two different genes and their associated circles (b) The new gene from crossover and the new circle. The defining minimal subset points are indicated by the arrows.

This is where the GA comes in, it uses the crossover operator to improve these local estimates. First, the GA chooses two primitives to mate randomly, with their selection probability biased by their fitness function values. Once two primitives are chosen to mate, the crossover operation is applied to create two new primitives. As we have discussed previously crossover is the key component in the GA algorithm. For a GA to work well, partial solutions (chromosome substrings) must be used as building blocks which are combined to create better solutions [Goldberg, 1988]. In our case these partial solutions are the local estimates of the primitive in the initial population. The crossover operation does indeed combine them to produce primitives which are often better than either parent. Figure 5.2 shows how crossover accomplishes this task. In this figure, it is assumed that all the points belong to a single circle. In part (a) of the figure are two circles, described by two different chromosomes, each consisting of three points. The points that make up the chromosome of the first circle are shaded in black, the second circle are shaded in grey and the remaining points are not shaded. Assume that both chromosomes were part of the initial population obtained by choosing geometric data points over local portions of the image, which explains why the minimal subset points of the chromosome are close to each other. It is clear from the figure that while each chromosome is a good estimate of a local

portion of the circle, neither chromosome defines a circle which is a good description of all of the points that make up the complete circle. Crossover has the potential to improve on this estimate by combining the two chromosomes. In part (b) of the figure, one of the possible children produced by the crossover operator is shown. The new chromosome points are the three shaded points (two black and one grey) marked by the arrows. The crossover operation has spread the minimal subset across more of the geometric data. As is the case here, this spreading often produces a better primitive than the one defined by either of the parents. In this case better means a primitive which fits more of the points on the curve or surface. This demonstrates that the crossover operator is indeed able to use partial solutions (chromosome substrings) as building blocks to produce better global solutions.

It is important to note that crossover is much more likely to produce a better geometric primitive when the two parents are similar to each other. In other words a crossover operation applied to two widely differing circles is much less likely to produce a better circle than either of the parents. For this reason we augment the selection process by using a compatibility filter. This chooses two parents to mate in the normal GA fashion, but disallows the mating if the parents fail to pass a compatibility test [Goldberg, 1988]. The reasoning is that the children of incompatible population members are unlikely to be fitter than their parents. While this is not true in all applications, it is the case for primitive extraction. The compatibility function measures the similarity of the parents by computing the average distance of the minimal subset points of one parent to the curve or surface defined by the other parent. The more similar the two primitives, the less this average distance, and the more compatible the parents. For identical primitives this compatibility function returns zero, and its value increases as the primitives differ. According to this measure the two circles shown in Figure 5.2 are compatible. This means that performing crossover on the minimal subset chromosomes of these two circles has a good chance of producing children who are fitter than their parents. Whether the two parents can mate is decided by thresholding this compatibility function. In this way compatibility checking speeds up convergence of the GA by only allowing mating of

compatible parents.

The fitness function is applied to the two children produced by the crossover operation. The two primitives with the smallest cost function values (the least fit) are then removed from the population pool to keep it a fixed size. The basic GA loop is repeated until there is some indication of convergence. In our experiments this is indicated when the best population member has not changed in fifty mating operations. Experiments show that convergence is very fast, in the order of a few hundred iterations. The theoretical reason is that an exponentially increasing amount of effort is afforded to the dominant population members, as was explained in the previous section [Goldberg, 1988].

The population member with the best score is the final result of this process. The chromosome of this individual completely defines the geometric primitive since the parameter vector $\overline{a}$ can be obtained from it, as was shown in Chapter Three. This parameter vector, along with the definition of the cost function, enables each of the geometric data points that belong to the primitive to be labelled using the procedure described in the Chapter Two. In order to extract all the primitives, the genetic algorithm is run repeatedly on the remaining geometric data points not on any primitive. This procedure is repeated until there are too few geometric data points remaining for a significant primitive to be extracted.

5.4 Experimental Results

There is a number of parameters that must be set in order to use this algorithm, and we will now discuss how they are chosen. As discussed in the previous section, the cost or fitness function we use counts the number of data points with a small template around the curve or surface. The first parameter is the size of this template. For the circle and ellipse extraction examples that follow, the template size is set to one pixel, while for the plane and sphere extraction the template size is set to one millimeter. We set the template size to the accuracy of the geometric data, which was estimated by studying the characteristics of the sensor used to create the data.

The difference in units is due to the fact that different sensors were used to create the geometric data in different examples. The next important parameter is the size of the initial population. Experience has shown that the initial GA population should be at least thirty and at most a few hundred geometric primitives [Goldberg, 1988]. In our examples, we demonstrate three different ways of creating this initial population. For the circle extraction example we use a k-d tree, for the ellipse extraction we use connected chains, and for the plane and sphere extraction we divide the data into small windows. In all these cases the parameters (k-d tree size, chain size, and window size) were set to give an initial population of at least thirty primitives. If the initial population of the GA is lower then there is a significant chance that some primitives will be missed.

The first examples have as input the edge points extracted from an intensity image produced by a digitizer. These are places where there is a significant image discontinuity, and they consist of an unordered list of 2D points. They were created using a public domain image processing package with the default threshold settings. No attempt was made to optimize the results in any way by experimenting with these settings. We demonstrate the extraction of circles and ellipses from edge data in which there are a significant number of such primitives. For some examples the edge points have been divided into chains, where a chain is a connected set of such points [Giraudon, 1987]. The initial population for the GA is then created by the random sampling of minimal subsets on each of these connected chains. Since a chain is likely to be a part of only a single primitive, this method of creating the initial population is very effective. For other examples the edge points have been partitioned by using a k-d tree [Samet, 1984]. This is a recursive data structure which is created by subdividing the image until each terminal node of the k-d tree contains no more than a certain number of points. For a hierarchic data structure a terminal node is the lowest level of this structure. Again the initial population is created by random sampling of minimal subsets, but this time using the edge points that belong to each terminal node of the k-d tree as the input.

The first image is of a number of coins, and the task is to extract the circles defined

by these coins. In Figure 5.3, part (a) is the original intensity image of the coins. In part (b) of this figure are the edge pixels produced by the image processing package. Note that there are gaps in the circles, along with some spurious points produced by the markings on the coins. Part (c) of this figure shows the k-d tree decomposition of the edge points, with each of the smallest boxes defining the terminal nodes of the tree that contribute to the initial population. The points that make up the initial population are obtained by random sampling of minimal subsets on these terminal nodes, and are shown in part (d). Part (e) of this figure shows the circles that have been extracted by the GA, and they are superimposed on the original edge points in part (f). The circles were obtained by repeatedly running the GA, removing the points that belong to the highest scoring population member, and stopping when the number of remaining edge points is below a threshold.

The rapid convergence of the GA for circle extraction is demonstrated in Figure 5.4. In this figure all the minimal subset points for the circle extraction problem are drawn as small dots. The best current circle in the population is drawn as three large dots. Part (a) of this figure shows the initial population, while parts (b) through (d) show how the population evolves as the GA executes. Each of these figures is separated by forty crossover operations. In a very short number of iterations the GA population has converged to a single answer. Note that in part (b) the best circle is different than the one in part (a) as a result of the crossover operator. This circle remains the best one until the convergence of the GA in part (d). Once the entire population is identical, convergence has definitely occurred; however, the GA need not be run until this point. Usually it is run until the best population member has not changed in a given number of iterations.

Instead of using the GA mechanism of random selection by fitness, it is possible to always combine the two best primitives in the population. Such an algorithm would have the same basic structure as the GA, but be completely deterministic. It would resemble a traditional computer-vision grouping algorithm [Levine, 1985]. However, such grouping algorithms commonly suffer from the problem of premature commitment which occurs when a particular solution is chosen too early in the search

process. The underlying cause is the deterministic control structure in which the two best primitives are repeatedly combined. The best geometric primitive is not necessarily produced in this way. An example of this situation is shown in figure 5.5. In part (a) of this figure are two circles. The one on the left has two disconnected parts, and the one on the right has four. Even though each part of the smaller circle contains more points than any single part of the larger circle, the total number of points on the larger circle is greater than the smaller. Thus the larger circle is the better solution, and is the one that should be found first by the extraction process. Assume that the initial population was created by random sampling on connected chains. A grouping algorithm that combined the two best circles at each iteration would extract the smaller circle. This is because initially the two longest chains, and therefore the two best circles, are on the smaller circle. These would be merged together, and the grouping algorithm would have prematurely found a sub-optimal solution.

The GA by contrast finds the larger circle. The initial population of the GA are the small dots in part (b), and the chromosome of the best circle is defined by the three large dots. It was created by random sampling on the connected chains of 2D edge points. In the initial population the best chromosome is on the smaller circle since this chain contains the most points. However, after a number of iterations the best chromosome is on the larger circle because crossover has created a chromosome that spans the various parts of the circle. Similarly in part (c) the population after ten iterations of the GA is shown, along with the chromosome defining the best circle. Premature convergence was avoided because the selection process of the GA still allocated some effort to the parts of the larger circle, even when the current best population member was on the smaller circle. This is because the GA selection process is not deterministic. Instead, the GA explores a number of possible alternatives in parallel and slowly evolves to what it considers to be the best solution. As time goes on, the diversity of the population decreases, but this happens slowly. If the initial population is too small then premature convergence can occur; if too large, convergence takes a long time. If the population size is reasonable (between fifty

and three hundred members) then the GA method usually does not suffer from the problem of premature commitment and still converges quickly.

In our experiments an extraction procedure that relies only on random search is noticeably slower than the GA. Further experiments must be done to quantify the speedup. We can say that the difference between a GA and random search is significant when there are many primitives to extract or the primitives are complex, that is they have large minimal subsets. This is not surprising, since we have shown in Chapter Two that in these cases many random samples are necessary for successful extraction [Rousseeuw and Leroy, 1987]. Thus the more complex the scene, the better the GA will perform in comparison to random search. When a single geometric primitive contains a significant percentage (more than 30%) of all the data points the GA extraction algorithm is not noticeably faster than random search.

An example of the flexibility of the GA approach is shown in Figure 5.6, where instead of extracting circles, we extract two ellipses from an image of a cable surrounded by an insulator. The original intensity image was shown in Figure 1.1 in Chapter One. In this case the initial population is found by random sampling over a chain, which is a connected set of edge points. While using chains produces better results than a k-d tree, they have the disadvantage of taking longer to create. In part (a) of this figure are shown the initial chains. Parts (b) and (c) show the two extracted ellipses, along with the minimal subset points that define each ellipse drawn as black dots. The fact that the two ellipses are very close together makes this a particularly difficult example. Since the ellipse has five degrees of freedom, an extraction algorithm that used only random sampling approach would require an excessive number of samples. The only change necessary to the GA algorithm in order to extract ellipses instead of circles was to use a chromosome with five points, instead of three.

The last example has as input a number of three-dimensional points produced by a laser rangefinder mounted on a robot wrist [Rioux and Blais, 1986]. This rangefinder collects parallel profiles, where the number of profiles and the spacing between them is controllable. In practice, such data are relatively sparse since the spacing between profiles is usually fairly large. This example demonstrates the extraction of a sphere

from such laser rangefinder data. The initial data is shown in Figure 5.7, part (a), with the sphere being the topmost object. The white area below the points on the sphere is the area where the laser beam is blocked by the sphere. Part (b) of this figure shows the points belonging to the extracted sphere shaded in black. The same GA algorithm was used as in the 2D examples, except that the initial population was created by random sampling on ten by ten windows spread over all the geometric data.

The main conclusion we have drawn from these experiments is that the quality of the results depends a great deal on the quality of the initial population. This in turn, depends primarily on how the initial population is obtained. For example, when using k-d trees most of the circle extraction examples are successful; however, ellipse extraction is less successful. This is because ellipse extraction is inherently more difficult than circle extraction since the size of the minimal subset is five for an ellipse, versus three for a circle. Thus a better initial population is necessary for extracting ellipses, which is provided by the use of chains. For the three-dimensional data we have found that using small windows to create the initial population gives reliable plane and sphere extraction. This is because the three-dimensional laser rangefinder data is accurate enough to make the initial population members obtained in such small windows reasonably accurate. The more difficult the extraction problem, the more care that must be taken in creating the initial population. This is not surprising, since it is the points in the initial population that the GA uses to make new population members.

5.5 Summary

The GA succeeds because it uses the local estimates of the curve or surface provided by the initial population as a building block to find a better global estimate. It requires a good chromosome representation to function properly. In our case, this representation encodes a geometric primitive by the minimal set of points necessary to define the primitive uniquely. This is much more efficient than coding the parameters

of the primitive as a bit string. It is the same representation we have used for our random sampling extraction algorithm. It has not, to our knowledge been used in a GA. We have shown examples of circle, ellipse and plane extraction, but can also extract spheres, cylinders and ellipse from three-dimensional data using the same approach. In fact, the GA algorithm can be used to extract the same primitives as the basic random sampling algorithm. All that is necessary is that the chromosome representation be transformed into a parameter vector, which can be done for many different types of primitives. We have not shown how to use the GA for correspondence computation, which is a future area of research.

Genetic algorithms have rarely been employed in the computer vision field (for two recent exceptions see [Bhanu *et al.*, 1991, Hill and Taylor, 1992]). This is surprising since the use of simulated annealing, a closely related technique, is widespread [Geman and Geman, 84, Corana *et al.*, 1987]. Once it is understood that primitive extraction is actually an optimization problem, the use of a GA suggests itself. We have shown that using a GA is a very effective approach to solving the primitive extraction problem. Since it is based on the evolutionary metaphor it often avoids the problem of premature commitment. Instead, the best solution emerges spontaneously from the population over time.

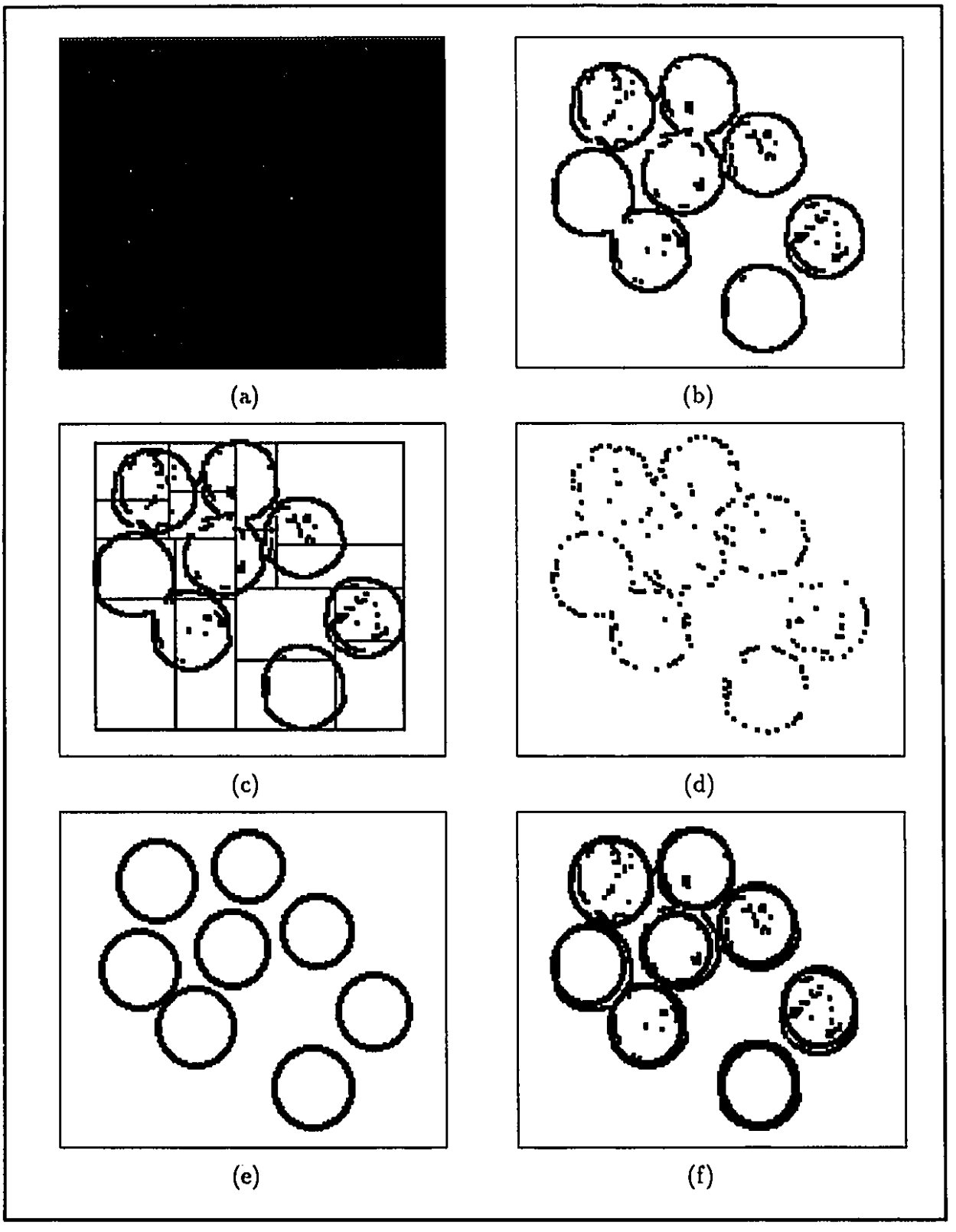

Figure 5.3: Extracting circles from a complex image (a) The original image (b) The edge pixels (c) The k-d tree used to create the starting population (d) The points in the initial population (e) The extracted circles (f) The extracted circles superimposed on the edge pixels

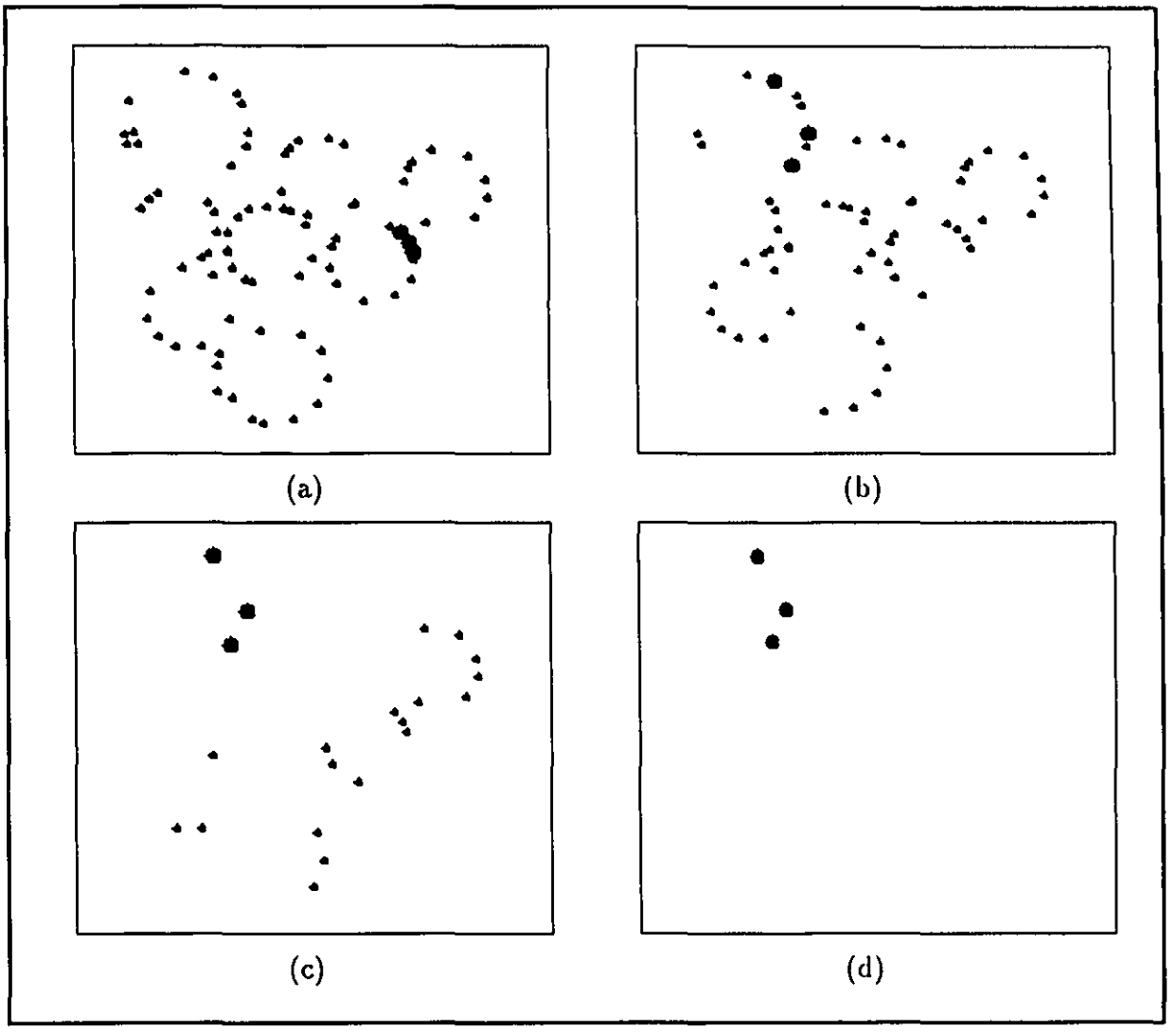

Figure 5.4: Population of a GA during execution of circle extraction. The points defining the best circle in the population are drawn as three large dots. (a) The original population (b) After 50 crossovers and mutations (c) After 90 crossovers and mutations (d) After 130 crossovers and mutations

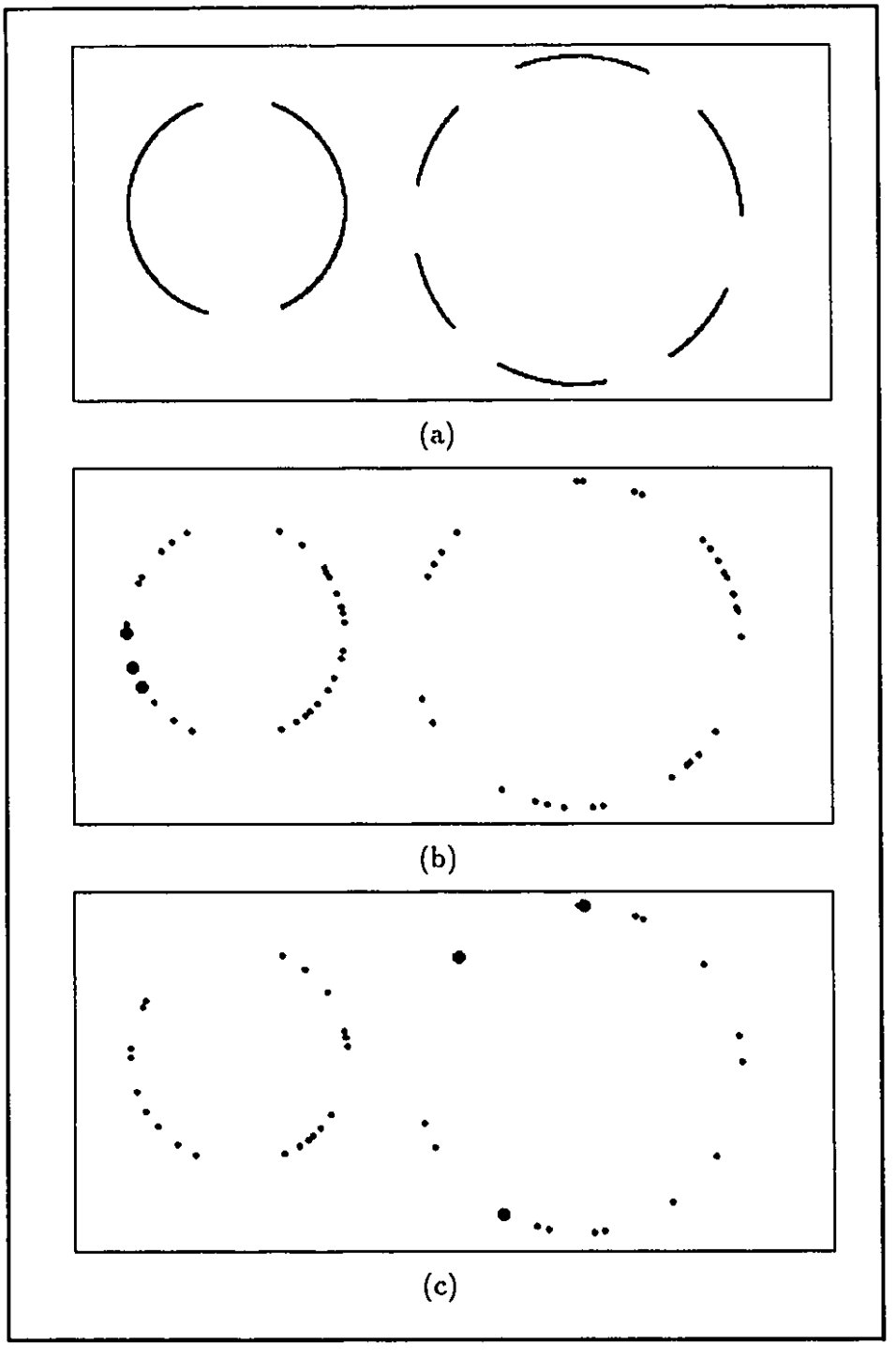

Figure 5.5: An example of how a GA avoids premature convergence. The points defining the best circle in the population are drawn as three large dots. (a) two circles (b) the initial GA population and best member (c) the population and best member after a number of crossover and mutation operations

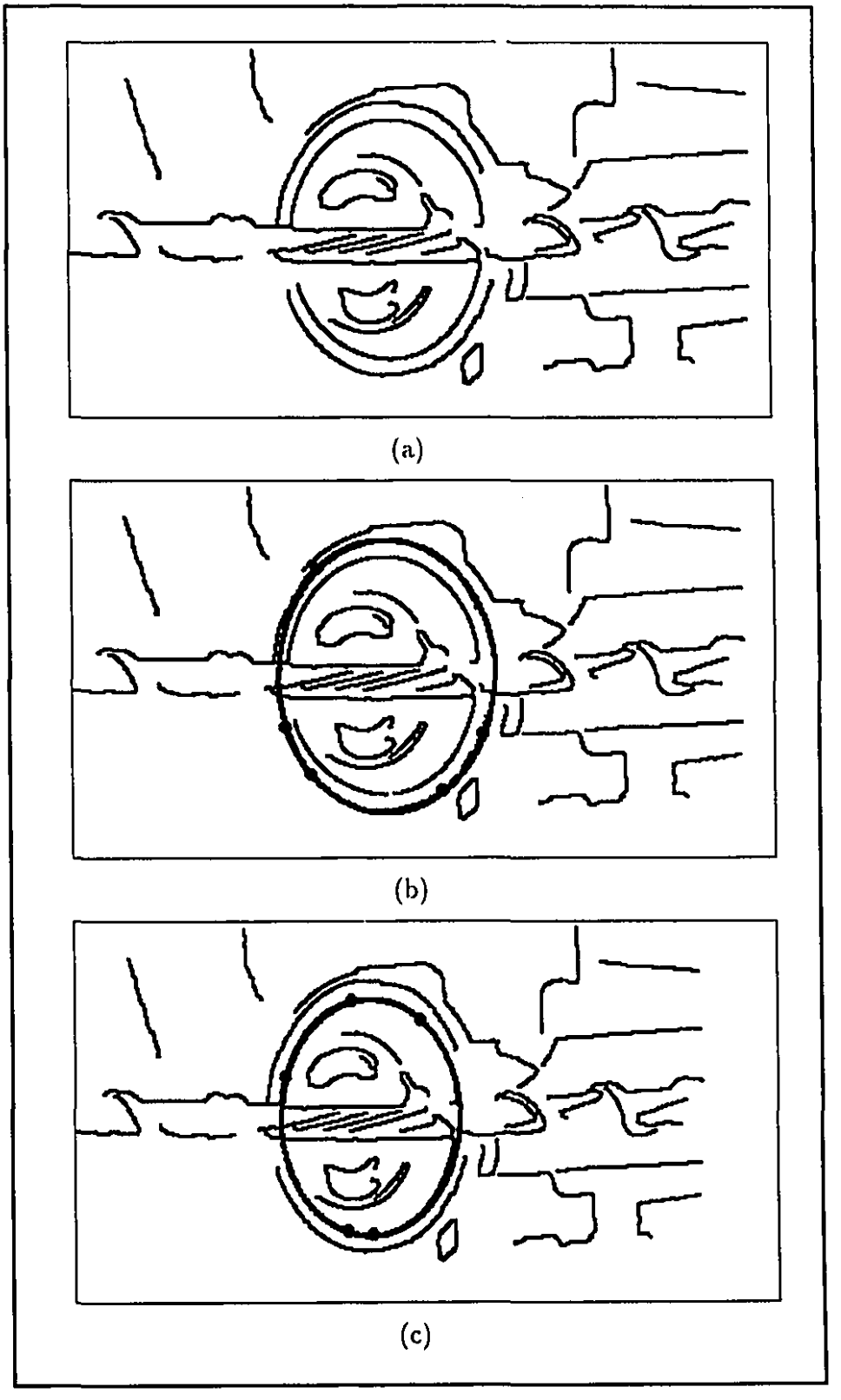

Figure 5.6: Extracting ellipses from cable image (a) initial image (b) first extracted ellipse (c) second extracted ellipse

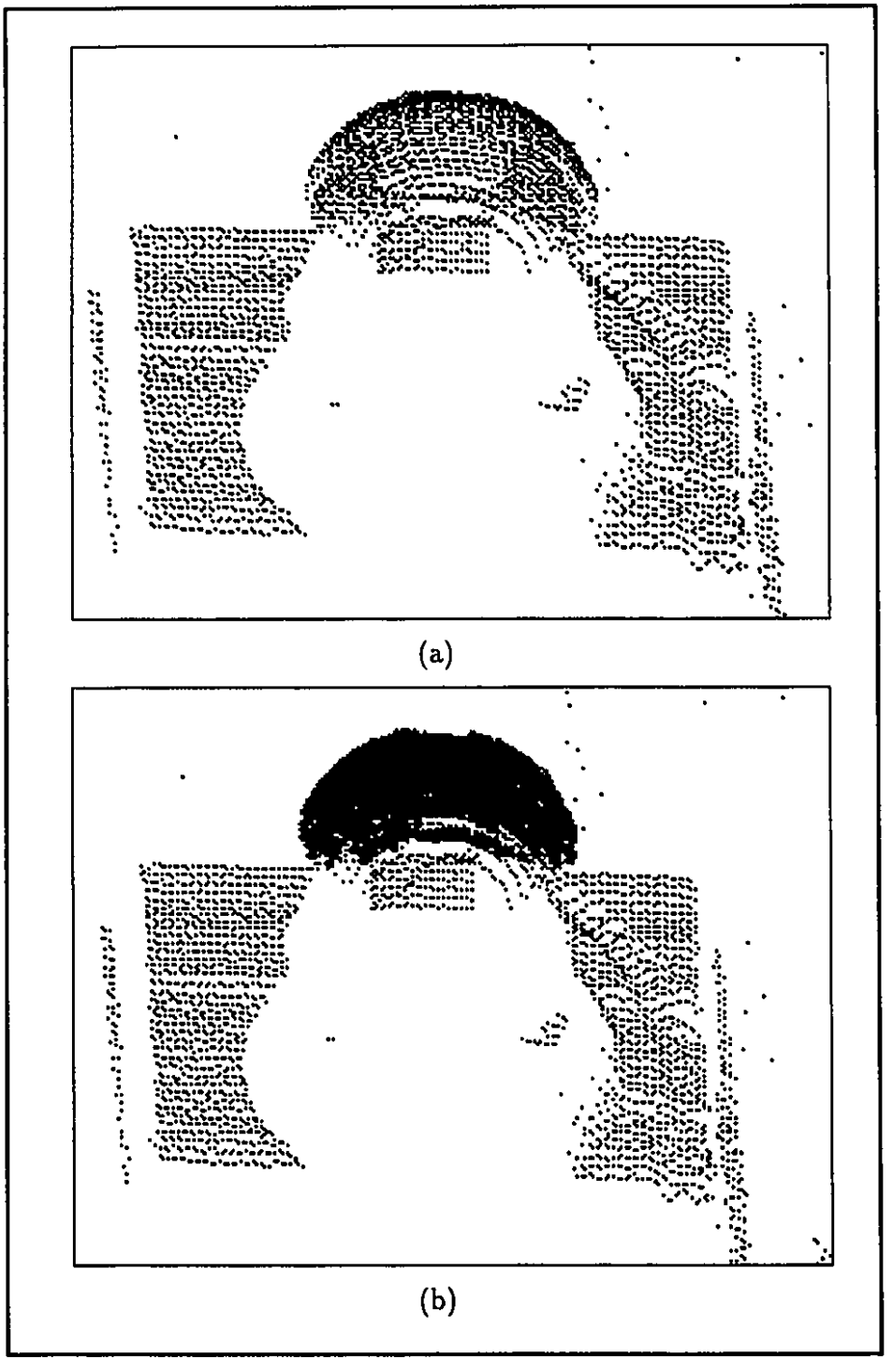

Figure 5.7: Sphere extraction from range data (a) initial range data with sphere on top (b) extracted sphere points shaded in black.

Chapter 6 Parallel Implementations of Extraction

It is important to be able to parallelize an extraction algorithm. One reason is that parallel architectures are becoming more widely available. Another reason is that as shown by our optimization model, extraction is inherently a computationally intensive problem. Parallel architectures will be necessary when processing a large amount of geometric data, or when attempting to achieve real-time performance. There are many different types of parallel architectures, and it is not yet clear that any single one will be used to the exclusion of others. Thus it is necessary to consider the problem of parallelization for a wide variety of architectures.

In this chapter we will discuss ways of parallelizing the primitive extraction algorithm. A large percentage of the execution time of both the GA and the random sampling version of this algorithm is spent in the evaluation of the cost function. If there are N geometric data points, then this evaluation takes $O(N)$ time for most cost functions. For any significant sized N this is the most expensive part of the algorithm in computational terms. Therefore we concentrate our efforts on parallelizing this cost function evaluation. In the ideal case this should be done in $O(1)$, as opposed to $O(N)$ time. We will show that, depending on the architecture, this ideal can indeed be achieved. The less time taken in cost function evaluation, the more evaluations that can be done per unit time, and the better the extraction algorithm will work.

In this chapter we will discuss ways of parallelizing the fixed-band cost function, and the variable-band cost function evaluation for primitive extraction. Since most of our examples of Chapter Two were processed using the fixed-band cost function, we will naturally concentrate on it's parallelization. The result returned by any fixed-band cost function is the sum of a simple function applied to each data point. In other words, if there are Ngeometric data points then $h = \sum_{i=1}^{i=N} g(r_i)$, where g is the simple cost function applied to each data point. The most common simple cost

function g, is equivalent to template matching. However, this is not the same as traditional template matching. In traditional template matching a template is a two-dimensional array which is applied to an image, that is itself a larger two-dimensional array [Levine, 1985]. This makes the template matching a discrete cross correlation procedure. Algorithms which claim to implement template matching on parallel hardware are based on this description of template matching [Kumar and Krishnan, 1989]. In our case the templates are fixed size areas around a given geometric primitive. The shape of such a template changes with the primitive. Therefore, this template cannot be matched by a simple convolution process since the template shape changes.

Our parallelization algorithm approach also differs from other approaches applied to parallelizing genetic algorithms (GAs) [Ackley, 1987, Tanese, 1989, Manderick and Spiessens, 1989]. To parallelize a GA each processor usually executes the entire GA program, including the evaluation of the cost (or fitness) function. For many traditional GA applications the cost function evaluation is very cheap computationally, so this approach makes sense. In our situation this is not true since this cost function evaluation stage dominates the computation time of the GA, which explains our attention to it.

Parallelizing only the cost function evaluation has two other important advantages. First, it is possible to accomplish this task on a wide variety of different architectures. If the entire extraction algorithm were to be parallelized this would be more difficult to achieve. Second, this speeds up both the random sampling, and the GA version of the extraction algorithm. If we attempted to parallelize the entire extraction algorithm it would be necessary to use different approaches for the random sampling and the GA versions. In this chapter we will describe parallel implementations of the cost function of the extraction algorithm on a variety of different architectures. We will start with the simplest architectures, and move on to more complex ones.

6.1 Vector Architectures

Some of the earliest parallel architectures were vector machines [Duncan, 1990]. These had as their basic data structure a vector, whose elements could be operated on in parallel. While superseded by multi-processor architectures there are still many vector machines in operation. Recently the vector model has been revived [Little *et al.*, 1989]. The justification for this revival is two-fold. First, it is now possible to map vector operations to a number of different multi-processor architectures. Thus, a description of an algorithm as a set of vector operations is in some sense architecture-independent. Second, the common and inexpensive Digital Signal Processor (DSP) can be thought of as a vector machine. These DSP's can be easily cascaded together to perform vector operations very quickly. We will describe a model that defines a set of operations on vectors, and return either a vector or a scalar as their output. Then the cost function evaluation is described in terms of these abstract vector operations. To make these programs run on a specific vector architecture it is necessary to map these abstract vector operations to this architecture. Since these abstract vector operations are relatively simple this is not a difficult task. The following are the abstract vector operations that we need for our algorithm:

- Binary Operations $(+, -, /, *)$
- Sort
- Count
- Index
- Dot Product

A more detailed description of each of these operations is given below.

1. Arithmetic - takes two vectors as input and outputs a vector which is the pairwise arithmetic operation applied to each of the elements, for example:

$$A = [51343926]$$

$$B = [25381362]$$

$$A + B = [7661241288]$$

$$A * B = [1059323271212]$$

2. Sort - takes a single vector as input and outputs a vector, with the individual elements sorted in order from the smallest to the largest, for example:

$$A = [51343926]$$

$$Sort(A) = [12334569]$$

3. Count - takes a single vector and a scalar as input and outputs a scalar which is a count of the number of vector elements less than or equal to the input scalar, for example:

$$A = [51343926]$$

$$Count(A, 3) = 4$$

4. Index - takes a single vector and a scalar as input and outputs the Nth element of the first vector, for example:

$$A = [51343926]$$

$$Constant = 5$$

$$Index(A, Constant) = 3$$

5. Dot Product - takes two vectors as input and outputs a scalar which is their dot product, for example:

$$A = [51343926]$$

$$B = [25381362]$$

$$A \bullet B = 110$$

With only these simple vector operations it is possible to parallelize both the fixed-band, and variable-band scoring functions for primitive extraction.

First we will show how to parallelize the computation of the distances of each point to a particular geometric primitive. The method we describe will work for geometric primitives which are defined as a linear combinations of basis vectors. For this types of primitive the implicit function f can be written as $f(\overline{p};\overline{a}) = \overline{a} \bullet \overline{b}$, where $\overline{a}$ is the coefficient vector, and $\overline{b}$ is the basis vector. For example, consider the case where the primitive is a line. Then the basis vector is $(1, x, y)$ and the coefficient vector is a_0, a_1, a_2. If there are N geometric data points then $f(x_1, y_1) = a_0 + a_1 x_1 + a_2 y_1$, $f(x_2, y_2) = a_0 + a_1 x_2 + a_2 y_2$, and so on till $f(x_N, y_N) = a_0 + a_1 x_N + a_2 y_N$. This set of equations can be rewritten as a sequence of operations on vectors of dimension N. In our notation we take $\overline{a_0}$ to be a vector of dimension N with N identical entries; that is, $\overline{a_0}$ is $[a_0, \ldots, a_0]$. Similarly $\overline{x} = [x_1, \ldots, x_N]$ and $\overline{y} = [y_1, \ldots, y_N]$. Let $\overline{funcvect} = [f_1^2, \ldots, f_N^2]$ be an N-dimensional vector whose elements are the values of the implicit function f evaluated at each geometric data point. For the case of a line $\overline{funcvect} = \overline{a_0} + \overline{a_1} * \overline{x} + \overline{a_2} * \overline{y}$. Thus $\overline{funcvect}$ can be computed by the operations of our vector machine, using only $+$ and $*$, the abstract vector operations described above. Similarly $\overline{gradvect} = [|\nabla f_1|^2, \ldots, |\nabla f_N|^2]$ is an N-dimensional vector whose elements are the square of the magnitude of the gradient vector of f evaluated at each geometric data point. Similar arguments can be used to show that $\overline{gradvect}$ can also be computed by the operations of our vector machine in the same way as $\overline{funcvect}$.

Given the fact that the vectors $\overline{funcvect}$ and $\overline{gradvect}$ are available the cost function evaluation is straightforward. Then $\overline{funcvect}/\overline{gradvect}$ is equal to $\overline{distvect}$, which is a vector of size N whose elements are the square of the approximate distance of each geometric data point to the geometric primitive using the approximation described in Chapter Two. Applying the count operation on this vector produces the score for fixed-band template match cost function. This equals $count(\overline{distvect}, \text{Fixed} - \text{Band} - \text{Size} - \text{Squared})$, where

Fixed – Band – Size – Squared is the square of the fixed-band distance. Similarly for variable-band scoring applying the sort operation followed by the Nth element returns the variable-band score. This equals $index(sort(dist), \text{Inlier} - \text{Fraction} \times N)$, where Inlier – Fraction is the variable-band parameter. These parameters are described in more detail in Chapter Two. From this discussion it is clear that when the geometric primitives are linear combinations of basis vectors the evaluation of the cost function using fixed-or variable-band scoring can be achieved by the operations of this abstract vector machine.

All the operations of this vector machine, except the sort, can be implemented in $O(1)$ time on the appropriate hardware. Thus for fixed-band scoring the evaluation of the cost function can be done in $O(1)$, as opposed to $O(N)$ time. The sort can be done in $O(\log N)$ time, so for variable-band scoring the cost function can be evaluated in $O(\log N)$ time. This of course does not take into account the time necessary to transfer the geometric data points into vector form. If the number of geometric data points N is very large, the required vectors may be so large that the hardware is not capable of performing these operations at the maximal speed. Nevertheless, the ease and efficiency of the implementation of our algorithm on a vector machine is clear.

6.2 Multi-Processor Architectures

The term multi-processor architecture describes a wide family of modern architectures whose components consist of multiple processors that are connected together in some fashion. Because of their diversity various attempts have been made to describe these architectures at a more abstract level, so that parallel algorithms could be written in a fashion that is independent of the particular hardware implementation. The most common of these descriptions is the Parallel Random Access Machine (PRAM) model [Vishkin, 1983]. In this model there is a number of processors, each with local memory along with some global memory. Each processor is capable of executing a program and can access local and global memory as shown in figure 6.1.

The bottleneck in the execution of a parallel program is usually in the access

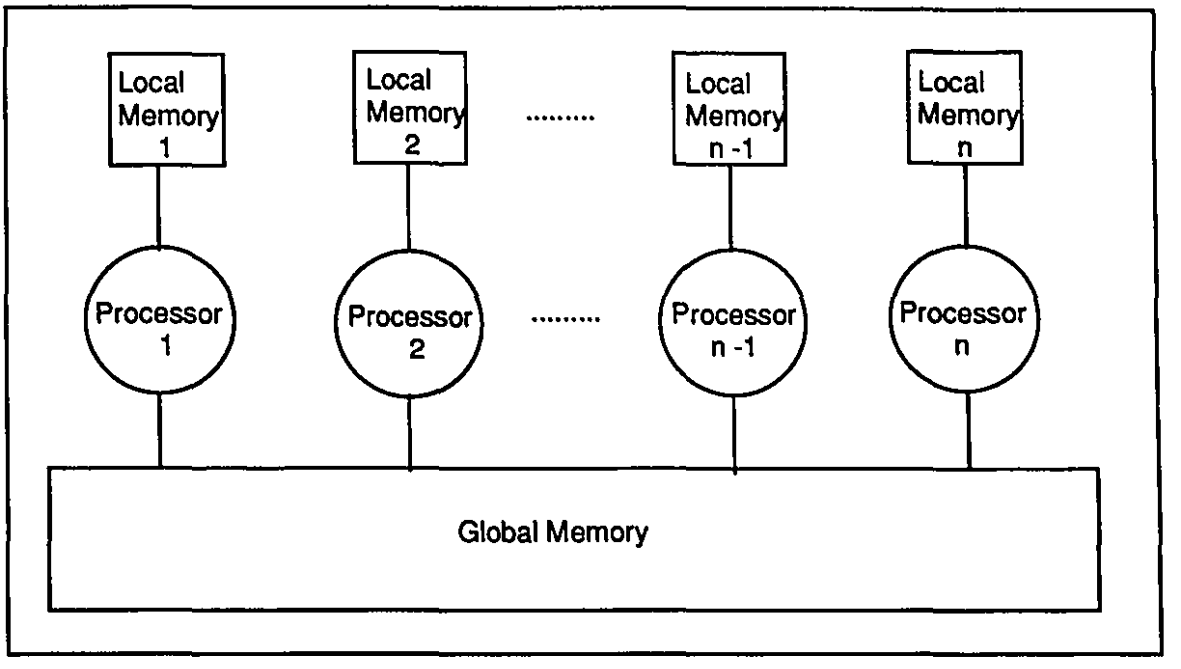

Figure 6.1: PRAM Model - Parallel Random Access Machine

of global memory. Assume that the N geometric data points were placed in global memory, and that each of the L processors were instructed to evaluate N/L of these points. Then at best there would be an L fold speedup in the cost function evaluation. However, in any physical multi-processor system there will be a memory access bottleneck if all the processors attempt to reference global memory. This fact is not easily represented by the PRAM models. While this model is useful theoretically, the level of abstraction is so high that many questions regarding its practical implementation on physical hardware are left unanswered.

Given this drawback of PRAM model the question remains as to whether there are any ways of characterizing different architectures. The following are what we believe are the basic dichotomies that can be used for this purpose.

- Coupling - Processors communicate with each other across communications channels. The degree of coupling between processors is measured by the bandwidth of the communications channel. A loosely coupled architecture has a low bandwidth between processors (such as a local area network) and a closely coupled architecture has a high bandwidth (such as those that share the same

address space on a common bus).

- Granularity - The number of processors is an indication of the granularity. In general, if there are fewer than a dozen processors then this is a coarse grained architecture, and if there from a dozen to a few dozen this is a medium grained architecture. When there are more than a few dozen processors the system is classified as a fine grained architecture.

- Instruction Stream - If each processor must execute the same instruction sequence then this is called an SIMD (single instruction, multiple data) machine. If on the other hand, each processor can execute different instruction sequences then this is an MIMD (multiple instruction, multiple data) machine.

- Topology - This the way in which the processors are connected to each other. There are many possible topologies, and some examples are the star (one processor at center with many wings), hypercube (three-dimensional cube), or systems that can implement arbitrary topologies such as the Connection Machine [Hillis, 1985].

Given these different ways to categorize a parallel architecture, the question is what is the most important characteristics of an algorithm for it to be easily parallelizable. We believe that it is that the algorithm should make as few demands on the inter-processor communication bandwidth as possible. The input should be partitioned into separate components which are given to each processor. The processors should operate independently on their portion of the data, and not communicate with each other during execution. If these requirements are not met then as the number of processors increases more and more time is spent on communications overhead, and the throughput does not increase proportionally. We claim that our primitive extraction algorithm has these characteristics, and can easily be parallelized on many different architectures. By contrast the HT is difficult to parallelize since its use of global memory during execution makes very high demands on the inter-processor communication bandwidth [Rosenfeld *et al.*, 1988].

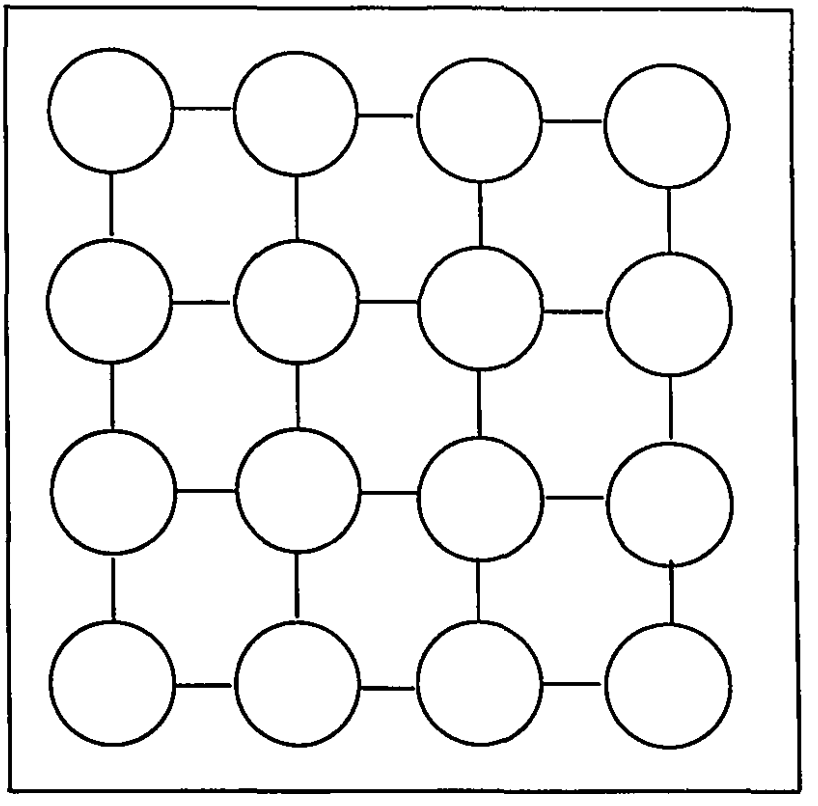

Figure 6.2: Mesh Topology Interconnection

For the multi-processor architectures we will only discuss the parallelization of the fixed-band cost function. This is the most common cost function, and was used in most of the examples in Chapter Two. We will describe a number of different categorizations of architectures using the above dichotomies. For each of these categories we will show how to parallelize the fixed-band cost function. Our list will not be exhaustive, but will cover what we consider to be the most important, and most common parallel architectures. The architectures are presented in increasing order of complexity.

6.2.1 Mesh Connected SIMD Architectures

This is probably the first, and still the most widely used multi-processor architecture for machine vision purposes. The mesh topology is shown in Figure 6.2. It is well suited to the processing of intensity images, with the usual approach being to allocate a single processor to each picture element (pixel) of the image. In the mesh topology

each processor can communicate directly with any of its neighbours in the mesh. This is ideal for performing local operations, such as convolutions, on an image. Since there are many pixels and therefore many processors, this is classified as a fine-grained architecture. Each processor in the mesh normally executes the same instruction sequence so meshes are usually SIMD (single instruction, multiple data) architectures.

The difficulty with a mesh topology is that it is not easy for processors to perform operations which require information from processors other than their neighbours on a mesh. Thus algorithms that require global access to memory, such as the HT, cannot be efficiently implemented on a mesh [Rosenfeld *et al.*, 1988]. However, because a mesh architecture is simple and relatively inexpensive it is still useful for many other computer vision algorithms. For this reason any algorithm which can be run efficiently on a mesh has the potential to achieve widespread usage.

We assume that the goal is to extract a 2D curve, and that edge detection has already taken place. Thus each edge element is associated with a particular processor on the mesh. In order to perform the fixed-band cost function evaluation for a geometric primitive its parameter vector must be broadcast to all the processors simultaneously. For the fixed-band scoring this can be done most efficiently if a global broadcast and collection facility is available on the mesh. Given the parameter vector those processors which have a valid edge point must decide whether their edge point is in the fixed-band. This would be done by calculating $f/\nabla f$ as described in Chapter Two, and computing the appropriate fixed-band cost function. Then the global collection function would simply sum the appropriate fixed-band cost function value computed by each processor. This sum is the resulting value of the fixed-band cost function for the primitive with the given parameter vector. If there is a processor associated with each pixel element then all these operations can occur simultaneously. Thus the cost function evaluation would require $O(1)$ instead of $O(N)$ time, where N is the number of geometric data points. Since each processor in the mesh already has an associated geometric data point no initialization stage in which these points are distributed among the processors is necessary. All that needs to be added to the

mesh architecture is a global broadcast and collection facility, which is not difficult.

It should be noted that this ability to perform fixed-band scoring is actually very useful for many model-based vision applications. A geometric model is nothing more than a list of geometric primitives. Often the presence of an object described by such a model has been hypothesized from some of the extracted geometric primitives. Then the presence of each primitive in the model can be verified against the geometric data by evaluating the cost function for each hypothesized primitive.

6.2.2 Medium and Coarse Grained Transputer-Like Architectures

Under this category are another common class of parallel architectures. There are architectures with anywhere from a half-dozen to a few dozen processors. These types of architectures often, but not always, have high bandwidth channels between processors, with each processor being of MIMD (multiple instruction, multiple data) type. Such architectures are more flexible than the SIMD type, but the price paid is an increase in the difficulty of programming. Since there are multiple instruction streams the synchronization of the processors must now be done in software, which requires advanced multi-processor operating systems [Gentleman *et al.*, 1987]. However, because of the limited success in using mesh connected SIMD architectures MIMD systems have become more common in the computer vision field. In particular, they have been used for implementations of the HT [Austin *et al.*, 1991, Ben-Tzvi *et al.*, 1989]. In terms of topology these systems vary, with a mesh and star being common. The degree of coupling between processors also varies. Some systems are very tightly coupled. Examples of this are situations where the processors occupy the same physical bus. In other situations, such as when processor communicate across an ethernet link, the coupling is loose.

Any fixed-band cost function can be evaluated in a distributed fashion by partitioning the geometric data equally among the processors. Then for a single evaluation of the cost function, each processor is given the value of the parameter vector, along

with the cost function definition. Each processor must then evaluate the cost function for each its subset of the geometric data. The sum of the answers from all the processors is the value of the cost function for all the geometric data. Once the geometric data points have been distributed, the number of data transfers necessary during execution of the algorithm is very small. Since this is the case, there is very little overhead as additional processors are added. Therefore the speedup should be proportional to the number of processors, which is the ideal.

This is essentially the same parallelization approach as was taken for the SIMD mesh architecture, with the only difference being that more than one geometric data point is assigned to each processor. However, because the processors in a MIMD system are more powerful and have more global memory than an SIMD system other approaches are also possible. One of these is to distribute all the geometric data points to all the processors at initialization. Then each processor could evaluate a different primitive than the other processors. This would happen if each processor were given a different parameter vector to evaluate, as opposed to sending the same parameter vector to all the processors. This makes minimal demands on interprocessor communication bandwidth, except for the initialization phase, when all the geometric data must be sent to each processor. This approach is similar to the parallel guessing strategy in which a number of processors interact via a blackboard [Fischler and Firschein, 1987]. However, if the amount of geometric data is large this approach is not practical, since the time taken to initially distribute the geometric data among the processors will be excessive. Thus we see that for more complex architectures cost function evaluation can be parallelized in a number of different ways.

6.2.3 Pyramid Architectures and Other Topologies

The most complex architectures such as the Connection Machine have a large number of small processors whose basic topology can be reconfigured [Hillis, 1985]. These processors may be SIMD, or MIMD machines. The only requirements our algorithm makes on the topology of the parallel architecture is a requirement for a global broad-

cast and collection facility. This can be implemented on many topologies, but the exact implementation does depend on the particular topology. A common topology for parallel architectures is a pyramid. In this architecture, as shown in figure 6.3, there is a hierarchy of processors, which can communicate with their descendents and predecessors in a pyramid. Recently it has been shown that a pyramid architecture can be implemented on a connection machine [Hillis, 1985]. A global broadcast and collection facility can be built into a pyramid architecture in a very natural and straightforward fashion. Once this facility is available the implementation of the fixed-band scoring is trivial.

We assume that each geometric data points is distributed to each of N processors at the lowest level of the pyramid. The function of the upper levels of the pyramid is to implement the global broadcast and collection facility. The top processor receives the parameter vector of the primitive to be evaluated. This is passed down to the processors at the next lowest level, and so on, to the lowest level processors. They each perform the cost function evaluation, and return the results upward. The top processor in the pyramid receives the value of the cost function for all the geometric data. Thus it is easy to compute the value of the fixed-band cost function using a pyramid architecture. Since pyramid architectures are very flexible, they are used for many other computer vision applications. Therefore as hardware costs decrease, such architectures will become more common.

6.3 Summary

In this section we have discussed how to implement the extraction algorithm on various parallel architectures. We have concentrated on speeding up the evaluation of the cost function since it is the most expensive computational component of our extraction algorithm for both the random sampling and the GA version. The fact that for many types of cost functions the data can be partitioned into sets that are operated on independently makes our algorithm simple to parallelize. The speedup should be proportional to the number of processors, since there is very little communication

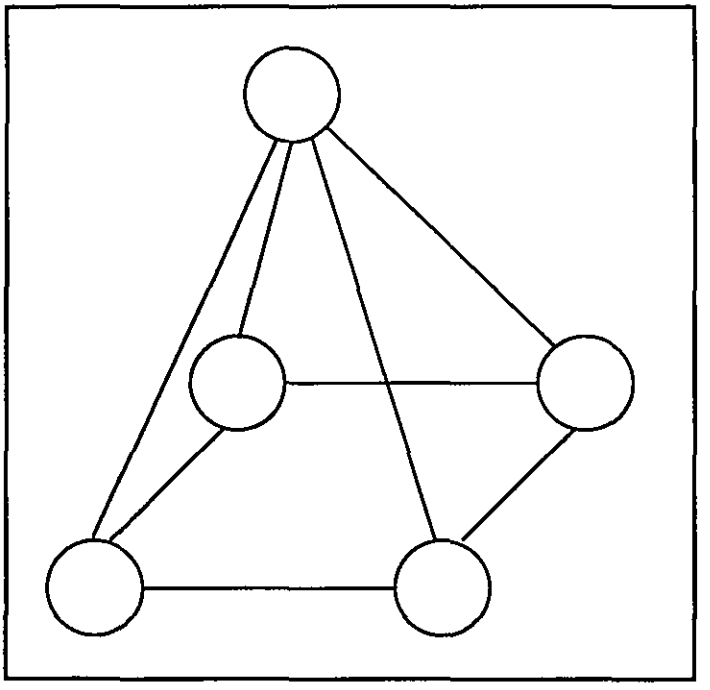

Figure 6.3: Pyramid Architecture Model

overhead necessary.

Chapter 7 Conclusion

This thesis has analysed the problem of primitive extraction in detail. In doing so we have described some new algorithms for solving this problem. We have also gained insight into the basic computational complexity of the problem of primitive extraction.

7.1 Contributions Revisited

In this research we have accomplished a number of things, and we summarize them below:

- We showed that primitive extraction is really an optimization problem.
- We generalized the solution to primitive extraction based on the random sampling of minimal subsets to apply to a much wider variety of curves and surfaces.
- We introduced a genetic algorithm, and showed how it can be applied to primitive extraction using the minimal subset representation.
- We showed how these algorithms can be parallelized on a variety of different parallel architectures.

This thesis has demonstrated that primitive extraction can be cast in an optimization framework. The goal is to find the global optimum of a cost function which usually has many local optima. A robust algorithm must be able to find a value close to the global optimum from among these local optima. Using this optimization model we obtain a deeper understanding of the complexity of this problem. We showed that primitive extraction is computationally difficult, and this difficulty is inherent in the problem definition.

The second thing we have done is to provide a general solution to the problem of primitive extraction based on minimal subsets. Minimal subsets are good representations when the geometric data are accurate. In fact, for perfectly accurate data

the random sampling of minimal subsets is probabilistically guaranteed to find the global optimum of the associated cost function. As the accuracy of the data decreases the probability of finding the global optimum also decreases. A key requirement of our approach is the ability to efficiently convert between a minimal subset and the parameter vector that describes the primitive or correspondence. We showed that using elimination theory, efficient, closed form solutions of these conversion equations can be generated. We compared our approach in detail with others in the literature, especially the Hough transform and the methods based on robust statistics.

We then introduced a genetic algorithm (GA) version of the primitive extraction algorithm. A GA is an approach to solving optimization problems which is based on an analogy with evolution. It enables the number of cost function evaluations, and thus the running time of the extraction algorithm to be substantially reduced. A GA represents individual solutions to the underlying optimization problem by means of a chromosome. Our chromosome representation is unique in that it is based on minimal subsets. A GA combines local solutions together using the crossover operator in a way that often avoids the problem of premature convergence. The best solution emerges spontaneously without a complex control structure guiding the algorithm. However, while the GA often converges to a good solution, there is no proof that it will converge to the global optimum. Thus, while it is often more efficient than random sampling it is still a heuristic approach.

The cost function evaluation is the most computationally intensive part of our algorithm. We showed that this can be parallelized on a wide variety of different parallel architectures. This is a significant advantage of our approach over other extraction algorithms. We described possible implementations on a wide range of parallel architectures.

The issue of robustness is central to our work. Our optimization model explains formally what robustness is, along with its computational cost. The model applies to any algorithm that attempts to solve the problem of primitive extraction. By dealing with the simplified case in which the geometric data have perfect accuracy, we gain an understanding of the basic complexity of this problem. For perfectly accurate

data random sampling is probabilistically guaranteed to find the optimum value of the cost function. The required number of random samples grows exponentially with the size of the minimal subset. However, for a fixed size minimal subset, the number of random samples necessary to extract the best primitive is a polynomial function of the number of geometric data points.

7.2 Future Work

There is a number of unexplored issues that need to be addressed. The first is to implement and test various parallel versions of our algorithms. Some preliminary work in this direction has already been done, [Meygret *et al.*, 1992], but further experimentation is necessary. There is as yet only a limited theoretical understanding of the GA. More work is necessary to quantify under what conditions it will converge to the global optimum. Further experimentation also needs to be done to quantify the difference between the GA and other stochastic search methods, such as simulated annealing. Another limiting factor is the inability to solve some of the minimal subset equations of certain geometric primitives using elimination theory. As more efficient versions of the Gröbner basis algorithm become available this problem should be alleviated. A related issue is how to apply this approach directly to curves and surfaces defined parametrically in order to create a robust version of the fitting algorithms for such curves and surfaces [Sarkar and Menq, 1992]. Another potential area of application for our algorithms are the problems of pose determination and refinement. Some preliminary work has already been done in this area [Roth and Levine, 1991c, Roth and Levine, 1992b], but more needs to be done.

7.3 Summary

Philosophically our approaches are members of a family of so called "weak methods" which are currently enjoying a resurgence in the field of Artificial Intelligence. Another member of this family are the connectionist algorithms typified by neural networks. Historically, these weak methods fell out of favour because of the inability

to extend them to more complex problems than the one for which they were originally developed. This produced the generally held hypothesis that the creation of intelligence requires a large amount of knowledge, and gave birth to the so called "strong methods" of Artificial Intelligence, sometimes referred to as the knowledge-based approaches. However, the difficulty of extracting and encoding knowledge, and the lack of robustness of the algorithms that use such knowledge, remain significant unsolved problems of the knowledge-based paradigm.

The problem of the lack of robustness in many machine vision algorithms has been noted in the research community [Sklansky, 1991]. For primitive extraction robustness is the ability to deal with outliers, and this requires the ability to find the global optimum, or a value close to it, of a cost function which has many local optima. Our main thesis is that robust and efficient solutions to primitive extraction can be obtained by simple algorithms running on fast hardware. Thus our contributions are three-fold. First, a deeper understanding of the problem of primitive extraction [Roth and Levine, 1990a, Roth and Levine, 1992a], second, our extensions of the solutions based on the random sampling of minimal subsets [Roth and Levine, 1990b, Roth and Levine, 1991b, Roth and Levine, 1991c, Roth and Levine, 1992d, Roth and Levine, In Press] and third, the use a genetic algorithm for solving the primitive extraction problem [Roth and Levine, 1991a, Roth and Levine, 1992c, Roth and Levine, 1992b].

Appendix A Solutions of Systems using Gröbner Bases

In Chapter Three there are a number of examples of algebraic systems that were solved by the Gröbner bases approach. In this appendix we describe in more detail how these solutions are obtained. Because of the length of the Gröbner basis produced by this method we will list only the last element of each basis. For this element there is only one unknown, which can be solved for using a closed-form or numerical solution. In all these examples the solution was found using the lexicographic ordering to produce the Gröbner basis on the Maple symbolic algebra system [Char *et al.*, 1988]. The number of equations in each of the Gröbner basis equals the number of unknowns in the system. With the lexicographic ordering the number of unknowns in the first basis equation is maximum, and decreases by one, till the last basis equation has only one unknown. Thus by solving the last equation first, and then repeatedly back substituting the solutions into the previous equations all the solutions of the system can be found.

A.1 Constrained Circle

$$\begin{aligned}(x_0 - C_x)^2 + (y_0 - C_y)^2 - r^2 \\ (x_1 - C_x)^2 + (y_1 - C_y)^2 - r^2\end{aligned} \tag{A.1}$$

The known variables are x_0, y_0, x_1, y_1, r and the unknowns are C_x, C_y. The last element of the basis is below, and it has one unknown, which is C_y.

$$\begin{aligned}&6\,x_1{}^2x_0{}^2 + 2\,x_1{}^2y_1{}^2 - 4\,x_1{}^2r^2 - 4\,x_1\,x_0{}^3 \\ &-4\,x_1\,x_0\,y_1{}^2 + 8\,x_1\,x_0\,r^2 + x_0{}^4 + 2\,x_0{}^2y_1{}^2 - 4\,x_0{}^2r^2 \\ &-4\,x_1{}^3x_0 + x_1{}^4 + 2\,x_1{}^2y_1{}^2 - 4\,y_1{}^2x_0\,x_1 + 2\,y_1{}^2x_0{}^2\end{aligned} \tag{A.2}$$

$$
\begin{aligned}
&+{y_1}^4 - 2\,{y_1}^2{y_1}^2 + {y_1}^4 \\
&+(8\,x_1\,y_1\,x_0 - 4\,y_1\,{x_1}^2 - 4\,y_1\,{x_0}^2 - 4\,{x_1}^2 y_1 \\
&-4\,{y_1}^3 + 4\,{y_1}^2 y_1 - 4\,{x_0}^2 y_1 \\
&+4\,{y_1}^2 y_1 - 4\,{y_1}^3 + 8\,y_1\,x_0\,x_1)Cy \\
&+(4\,{x_1}^2 - 8\,x_0\,x_1 + 4\,{x_0}^2 + 4\,{y_1}^2 \\
&-8\,y_1\,y_1 + 4\,{y_1}^2)Cy^2
\end{aligned}
$$

A.2 Constrained Ellipse

$$
\begin{aligned}
b^2(x_0 s_1 - y_0 s_2)^2 + a^2(x_0 s_2 + y_0 s_1)^2 - a^2 b^2 = 0 \\
b^2(x_1 s_1 - y_1 s_2)^2 + a^2(x_1 s_2 + y_1 s_1)^2 - a^2 b^2 = 0 \\
s_1^2 + s_2^2 - 1 = 0
\end{aligned}
\tag{A.3}
$$

The known variables are x_0,y_0,x_1,y_1 and the unknowns are s_1,s_2,a,b. The last element of the basis is below, and it has one unknown, which is b.

$$
\begin{aligned}
&{y_1}^4 a^4 {x_0}^4 - 4\,x_1\,{y_1}^3 y_0\,a^4 {x_0}^3 + 6\,{x_1}^2 {y_1}^2 a^4 {y_0}^2 {x_0}^2 - 4\,{y_0}^3 a^4 y_1\,{x_1}^3 x_0 \\
&+a^4 {y_0}^4 {x_1}^4 + (4\,{x_0}^4 {x_1}^2 {y_1}^2 a^2 + 2\,{x_0}^4 {y_1}^4 a^2 - 2\,{x_0}^4 {y_1}^2 a^4 - 8\,{x_0}^3 y_0\,y_1\,a^2 {x_1}^3 + 4\,{x_0}^3 a^4 y_0\,x_1\,y_1 \\
&+4\,{x_0}^2 {y_0}^2 {x_1}^4 a^2 - 4\,{x_0}^2 {y_0}^2 {y_1}^2 {x_1}^2 a^2 - 2\,{x_0}^2 {x_1}^2 {y_1}^2 a^4 \\
&-2\,{x_0}^2 a^4 {y_0}^2 {x_1}^2 - 2\,{x_0}^2 {y_1}^4 a^4 + 4\,{x_0}^2 {y_0}^2 {y_1}^4 a^2 \\
&-2\,{x_0}^2 a^4 {y_0}^2 {y_1}^2 + 4\,x_0\,y_0\,{x_1}^3 y_1\,a^4 + 4\,x_0\,y_0\,{y_1}^3 x_1\,a^4 - 8\,x_0\,{y_c}^3 {y_1}^3 a^2 x_1 \\
&+4\,x_0\,a^4 {y_0}^3 x_1\,y_1 + 2\,{y_0}^4 {x_1}^4 a^2 - 2\,{y_0}^2 {x_1}^4 a^4 \\
&-2\,{x_1}^2 {y_1}^2 a^4 {y_0}^2 + 4\,{y_1}^2 {y_0}^4 a^2 {x_1}^2 - 2\,a^4 {y_0}^4 {x_1}^2)b^2 \\
&+(4\,{x_0}^3 y_0\,a^2 x_1\,y_1 - 2\,{x_0}^2 {y_0}^2 a^2 {x_1}^2 + 4\,x_0\,y_1\,y_0\,a^2 {x_1}^3 \\
&+4\,x_0\,x_1\,y_0\,a^2 {y_1}^3 - 8\,x_0\,y_0\,a^4 y_1\,x_1 + 4\,x_0\,x_1\,y_1\,{y_0}^3 a^2 \\
&-2\,{x_0}^2 {x_1}^2 {y_1}^2 a^2 - 2\,{x_0}^2 {y_0}^2 a^2 {y_1}^2 - 2\,{x_0}^4 a^2 {y_1}^2 \\
&-2\,{x_0}^2 {y_1}^4 a^2 + 2\,{x_0}^2 a^4 {y_1}^2 + 2\,{x_0}^2 {y_0}^2 a^4
\end{aligned}
\tag{A.4}
$$

$$
\begin{aligned}
&-2\,x_0{}^2x_1{}^2a^4 + x_0{}^4a^4 - 2\,x_1{}^2y_0{}^4a^2 \\
&+x_1{}^4a^4 + a^4y_0{}^4 - 2\,a^2y_0{}^2x_1{}^4 + 2\,x_1{}^2a^4y_1{}^2 \\
&+2\,x_1{}^2y_0{}^2a^4 - 2\,x_1{}^2y_0{}^2a^2y_1{}^2 + y_1{}^4a^4 \\
&-2\,y_0{}^2a^4y_1{}^2 - 4\,x_0\,y_0{}^3x_1{}^3y_1 + 6\,x_0{}^2y_0{}^2x_1{}^2y_1{}^2 \\
&-4\,x_0{}^3y_0\,y_1{}^3x_1 + x_0{}^4y_1{}^4 + y_0{}^4x_1{}^4)b^4 = 0
\end{aligned}
$$

A.3 Three Dimensional Circle

$$
\begin{aligned}
a_0 + a_1x_0 + a_2y_0 + a_3z_0 &= 0 \\
a_0 + a_1x_1 + a_2y_1 + a_3z_1 &= 0 \\
a_0 + a_1x_2 + a_2y_2 + a_3z_2 &= 0 \\
(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2 &= 0 \\
(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2 &= 0 \\
(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2 &= 0 \\
a_0 + a_1x_c + a_2y_c + a_3z_c &= 0 \\
a_1^2 + a_2^2 + a_3^2 - 1 &= 0
\end{aligned}
\tag{A.5}
$$

The known variables are x_0,y_0,x_1,y_1,x_2,y_2 and the unknowns are $a_0,a_1,a_2,a_3,x_c,y_c,z_c$. The last element of the basis is below, and it has one unknown, which is z_c.

$$
\begin{aligned}
&y_2\,z_2\,y_1{}^3 - y_2{}^2y_1{}^2z_1 - x_2{}^2z_2\,y_1{}^2 \\
&-y_2{}^2z_2\,y_1{}^2 + y_2{}^3z_1\,y_1 - x_2{}^2y_1{}^2z_1 \\
&-x_1{}^2x_2{}^2z_1 - x_1{}^2y_2{}^2z_1 - z_2{}^3y_1{}^2 \\
&-y_2{}^2z_1{}^3 - x_2{}^2z_1{}^3 - x_1{}^2z_2{}^3 \\
&+y_2\,z_2\,y_1\,z_1{}^2 + x_2{}^2z_1\,y_1\,y_2 + z_2{}^2 \\
&z_1\,y_1\,y_2 + x_1\,x_2{}^3z_1 - x_1{}^2x_2{}^2z_2
\end{aligned}
$$

$$
\begin{aligned}
&-x_1{}^2y_2{}^2z_2 + x_1{}^3x_2\,z_2 \\
&+x_1{}^2y_2\,z_2\,y_1 + x_1\,y_1{}^2x_2\,z_2 + x_1\,y_2{}^2z_1\,x_2 \\
&+x_1\,z_2{}^2z_1\,x_2 + x_1\,x_2\,z_1{}^2z_2 + (2\,x_1{}^2y_2{}^2 \\
&-4\,x_1\,y_2\,y_1\,x_2 - 4\,y_2\,z_1\,y_1\,z_2 + 2\,y_2{}^2z_1{}^2 \\
&+2\,y_1{}^2x_2{}^2 + 2\,x_1{}^2z_2{}^2 + 2\,z_2{}^2y_1{}^2 \\
&-4\,x_1\,z_2\,z_1\,x_2 + 2\,z_1{}^2x_2{}^2)z_c
\end{aligned}
$$

A.4 Points to Lines

$$
\begin{aligned}
a_0 + a_1(x_0u_1 - y_0u_2 + h) + a_2(x_0u_2 + y_0u_1 + k) = 0 \\
b_0 + b_1(x_1u_1 - y_1u_2 + h) + b_2(x_1u_2 + y_1u_1 + k) = 0 \\
c_0 + c_1(x_2u_1 - y_2u_2 + h) + c_2(x_2u_2 + y_2u_1 + k) = 0 \\
u_1{}^2 + u_2{}^2 - 1 = 0
\end{aligned}
\tag{A.6}
$$

The known variables are x_0,y_0,x_1,y_1,x_2,y_2,a_0,a_1,a_2,b_0, b_1,b_2,c_0,c_1,c_2 and the unknowns are u_1, u_2, h, k. The last element of the basis is below, and it has one unknown, which is u_2.

$$
\begin{aligned}
&-b_1{}^2a_2{}^2y_2{}^2 - 2\,a_1\,c_0{}^2b_1 - a_1{}^2b_1{}^2x_1{}^2 \\
&-a_1{}^2a_2{}^2y_1{}^2 \\
&-a_1{}^2c_1{}^2x_2{}^2 - 2\,a_0\,c_1{}^2b_0 \\
&-b_1{}^2x_1{}^2c_1{}^2 - a_2{}^2y_1{}^2c_1{}^2 - a_1{}^2 \\
&a_2{}^2y_2{}^2 - 2\,a_1{}^2c_0\,b_0 - b_1{}^2c_1{}^2x_2{}^2 \\
&+4\,a_1\,c_1\,x_2\,b_1\,a_2\,y_2 \\
&+2\,a_1\,c_1{}^2x_2{}^2b_1 - 2\,a_1{}^2c_1\,x_2\,a_2\,y_2 \\
&+2\,a_1{}^2c_1\,x_2\,b_1\,x_1 - 2\,a_1\,c_1{}^2x_2\,b_1\,x_1 + 2\,a_1{}^2c_1\,x_2\,a_2\,y_1 \\
&-2\,a_1\,c_1{}^2x_2\,a_2\,y_1 + 2\,a_1\,a_2{}^2y_2{}^2b_1 + 2\,a_1{}^2a_2\,y_2\,b_1\,x_1
\end{aligned}
\tag{A.7}
$$

$$-2\,a_1\,a_2\,y_2\,b_1\,x_1\,c_1+2\,a_1{}^2a_2{}^2y_2\,y_1-2\,a_1\,a_2{}^2y_2\,y_1\,c_1$$

$$-2\,b_1{}^2c_1\,x_2\,a_2\,y_2-2\,b_1{}^2c_1\,x_2\,a_1\,x_1+2\,b_1{}^2c_1{}^2x_2\,x_1$$

$$-2\,b_1\,c_1\,x_2\,a_1\,a_2\,y_1+2\,b_1\,c_1{}^2x_2\,a_2\,y_1-2\,b_1{}^2a_2\,y_2\,a_1\,x_1$$

$$+2\,b_1{}^2a_2\,y_2\,x_1\,c_1-2\,b_1\,a_2{}^2y_2\,a_1\,y_1+2\,b_1\,a_2{}^2y_2\,y_1\,c_1$$

$$+2\,a_1\,b_1{}^2x_1{}^2c_1-2\,a_1{}^2b_1\,x_1\,a_2\,y_1+4\,a_1\,b_1\,x_1\,a_2\,y_1\,c_1$$

$$-2\,b_1\,x_1\,c_1{}^2a_2\,y_1+2\,a_1\,a_2{}^2y_1{}^2c_1-2\,a_0{}^2c_1\,b_1-2\,b_1{}^2c_0\,a_0$$

$$-2\,a_1\,b_0{}^2c_1+a_1{}^2c_0{}^2+a_0{}^2c_1{}^2$$

$$+b_1{}^2c_0{}^2+a_1{}^2b_0{}^2+b_0{}^2c_1{}^2+a_0{}^2b_1{}^2-2\,a_1\,c_0\,a_0\,c_1$$

$$+2\,a_1\,c_0\,b_0\,c_1+2\,a_1\,c_0\,a_0\,b_1+2\,a_0\,c_1\,b_1\,c_0$$

$$+2\,a_0\,c_1\,a_1\,b_0+2\,b_1\,c_0\,a_1\,b_0-2\,b_1\,c_0\,b_0\,c_1$$

$$-2\,a_1\,b_0\,a_0\,b_1+2\,b_0\,c_1\,a_0\,b_1+(2\,a_1{}^2c_0\,a_2\,x_2$$

$$-2\,a_1{}^2c_0\,a_2\,x_1-2\,a_1{}^2c_0\,c_1\,y_2+2\,a_1{}^2c_0\,b_1\,y_1$$

$$+2\,a_1\,c_0\,a_2\,x_1\,c_1-4\,a_1\,c_0\,b_1\,a_2\,x_2+4\,a_1\,c_0\,b_1\,c_1\,y_2$$

$$-2\,a_1\,c_0\,b_1\,y_1\,c_1-2\,a_0\,c_1\,a_1\,a_2\,x_2+2\,a_0\,c_1\,a_1\,a_2\,x_1$$

$$+2\,a_0\,c_1{}^2a_1\,y_2-2\,a_0\,c_1\,a_1\,b_1\,y_1-2\,a_0\,c_1{}^2a_2\,x_1$$

$$+2\,a_0\,c_1\,b_1\,a_2\,x_2-2\,a_0\,c_1{}^2b_1\,y_2+2\,a_0\,c_1{}^2b_1\,y_1$$

$$+2\,b_1\,c_0\,a_1\,a_2\,x_1-2\,b_1{}^2c_0\,a_1\,y_1-2\,b_1\,c_0\,a_2\,x_1\,c_1$$

$$+2\,b_1{}^2c_0\,a_2\,x_2-2\,b_1{}^2c_0\,c_1\,y_2+2\,b_1{}^2c_0\,y_1\,c_1$$

$$-2\,a_1{}^2b_0\,a_2\,x_2+2\,a_1{}^2b_0\,a_2\,x_1+2\,a_1{}^2b_0\,c_1\,y_2$$

$$-2\,a_1{}^2b_0\,b_1\,y_1-4\,a_1\,b_0\,a_2\,x_1\,c_1+2\,a_1\,b_0\,b_1\,a_2\,x_2$$

$$-2\,a_1\,b_0\,b_1\,c_1\,y_2+4\,a_1\,b_0\,b_1\,y_1\,c_1+2\,b_0\,c_1\,a_1\,a_2\,x_2$$

$$-2\,b_0\,c_1{}^2a_1\,y_2+2\,b_0\,c_1{}^2a_2\,x_1-2\,b_0\,c_1\,b_1\,a_2\,x_2$$

$$+2\,b_0\,c_1{}^2b_1\,y_2-2\,b_0\,c_1{}^2b_1\,y_1+2\,a_0\,b_1\,a_1\,a_2\,x_2$$

$$-2\,a_0\,b_1\,a_1\,a_2\,x_1-2\,a_0\,b_1\,a_1\,c_1\,y_2+2\,a_0\,b_1{}^2a_1\,y_1$$

$$+2\,a_0\,b_1\,a_2\,x_1\,c_1-2\,a_0\,b_1{}^2a_2\,x_2+2\,a_0\,b_1{}^2c_1\,y_2$$

$$-2\,a_0\,b_1{}^2y_1\,c_1)u_2+(b_1{}^2a_2{}^2y_2{}^2+a_1{}^2a_2{}^2x_2{}^2+a_1{}^2b_1{}^2x_1{}^2$$

$$
\begin{aligned}
&+a_2{}^2x_1{}^2c_1{}^2+b_1{}^2y_1{}^2c_1{}^2+a_1{}^2a_2{}^2y_1{}^2\\
&+a_1{}^2c_1{}^2x_2{}^2+a_1{}^2a_2{}^2x_1{}^2+a_1{}^2c_1{}^2y_2{}^2\\
&+b_1{}^2x_1{}^2c_1{}^2+a_2{}^2y_1{}^2c_1{}^2+b_1{}^2a_2{}^2x_2{}^2\\
&+a_1{}^2b_1{}^2y_1{}^2+a_1{}^2a_2{}^2y_2{}^2+b_1{}^2c_1{}^2x_2{}^2\\
&+b_1{}^2c_1{}^2y_2{}^2-2\,a_1\,c_1{}^2x_2{}^2b_1-2\,a_1{}^2c_1\,x_2\,b_1\,x_1\\
&+2\,a_1\,c_1{}^2x_2\,b_1\,x_1-2\,a_1{}^2c_1\,x_2\,a_2\,y_1+2\,a_1\,c_1{}^2x_2\,a_2\,y_1\\
&-2\,a_1\,a_2{}^2y_2{}^2b_1-2\,a_1{}^2a_2\,y_2\,b_1\,x_1-2\,a_1{}^2a_2{}^2y_2\,y_1\\
&+2\,a_1\,a_2{}^2y_2\,y_1\,c_1+2\,b_1{}^2c_1\,x_2\,a_1\,x_1-2\,b_1{}^2c_1{}^2x_2\,x_1\\
&-2\,b_1\,c_1{}^2x_2\,a_2\,y_1+2\,b_1{}^2a_2\,y_2\,a_1\,x_1-2\,b_1{}^2a_2\,y_2\,x_1\,c_1\\
&+2\,b_1\,a_2{}^2y_2\,a_1\,y_1-2\,b_1\,a_2{}^2y_2\,y_1\,c_1-2\,a_1\,b_1{}^2x_1{}^2c_1-2\,a_1\,a_2{}^2y_1{}^2c_1\\
&-2\,a_1{}^2a_2{}^2x_2\,x_1+2\,a_1{}^2a_2\,x_2\,b_1\,y_1+2\,a_1\,a_2{}^2x_2\,x_1\,c_1\\
&-2\,a_1\,a_2{}^2x_2{}^2b_1+2\,a_1{}^2a_2\,x_1\,c_1\,y_2-2\,a_1\,a_2{}^2\\
&x_1{}^2c_1+2\,a_1\,a_2{}^2x_1\,b_1\,x_2-2\,a_1{}^2c_1\,y_2\,b_1\,y_1\\
&-2\,a_1\,c_1{}^2y_2\,a_2\,x_1-2\,a_1\,c_1{}^2y_2{}^2b_1+2\,a_1\,c_1{}^2y_2\,b_1\,y_1\\
&-2\,a_1\,b_1{}^2y_1\,a_2\,x_2+2\,a_1\,b_1{}^2y_1\,c_1\,y_2-2\,a_1\,b_1{}^2y_1{}^2c_1\\
&-2\,a_2{}^2x_1\,c_1\,b_1\,x_2+2\,a_2\,x_1\,c_1{}^2b_1\,y_2+2\,b_1{}^2a_2\,x_2\,y_1\,c_1\\
&-2\,b_1{}^2c_1{}^2y_2\,y_1)u_2{}^2
\end{aligned}
\tag{A.8}
$$

A.5 Points to Line-Circle

$$
\begin{aligned}
(x_0+h)^2+(y_0+k)^2-r^2&=0\\
(x_1+h)^2+(y_1+k)^2-r^2&=0
\end{aligned}
\tag{A.9}
$$

The known variables are x_0,y_0,x_1,r and the unknowns are h,k,r. The last element of the basis is below, and it has one unknown, which is k.

$$
6\,x_0{}^2x_1{}^2+2\,x_0{}^2y_1{}^2-4\,x_0{}^2r^2 \tag{A.10}
$$

$$
\begin{aligned}
&-4\,x_0\,{x_1}^3 - 4\,x_0\,x_1\,{y_1}^2 + 8\,x_0\,x_1\,r^2 \\
&+{x_1}^4 + 2\,{x_1}^2{y_1}^2 - 4\,{x_1}^2r^2 \\
&-4\,{x_0}^3x_1 + {x_0}^4 + 2\,{x_0}^2{y_0}^2 \\
&-4\,{y_0}^2x_1\,x_0 + 2\,{y_0}^2{x_1}^2 + {y_0}^4 \\
&-2\,{y_0}^2{y_1}^2 + {y_1}^4 + (4\,y_1\,{x_0}^2 \\
&-8\,x_0\,y_1\,x_1 + 4\,y_1\,{x_1}^2 + 4\,{x_0}^2y_0 \\
&+4\,{y_0}^3 - 4\,{y_0}^2y_1 + 4\,{x_1}^2y_0 \\
&-4\,{y_1}^2y_0 + 4\,{y_1}^3 - 8\,y_0\,x_1\,x_0)k \\
&+(4\,{x_0}^2 - 8x_1\,x_0 + 4\,{x_1}^2 \\
&+4\,{y_0}^2 - 8\,y_0\,y_1 + 4\,{y_1}^2)k^2
\end{aligned}
\tag{A.11}
$$

$$
\begin{aligned}
(x_2u_1 - y_2u_2 + h) - r - p = 0 \\
u_1^2 + u_2^2 - 1 = 0
\end{aligned}
\tag{A.12}
$$

The known variables are x_2,y_2,r,p and the unknowns are u_1,u_2. The last element of the basis is below, and it has one unknown, which is u_2.

$$
\begin{aligned}
&-{x_2}^2 - 2\,x_2\,h - h^2 + r^2 + 2rp + p^2 \\
&+(2\,ry_2 + 2\,rk + 2\,py_2 + 2\,pk)u_2 \\
&+({x_2}^2 + 2\,x_2\,h + h^2 + {y_2}^2 + 2\,y_2\,k + k^2){u_2}^2
\end{aligned}
\tag{A.13}
$$

A.6 Three Dimensional Lines to Planes

$$
\begin{aligned}
(m_xs_1 - m_zs_2)w_1 + m_yw_2 = 0 \\
(n_xs_1 - n_zs_2)w_1 + n_yw_2 = 0
\end{aligned}
\tag{A.14}
$$

$$w_1^2 + w_2^1 - 1 = 0$$
$$s_1^2 + s_2^1 - 1 = 0$$

The known variables are m_x,m_y,m_z,n_x,n_y,n_z and the unknowns are s_1,s_2, w_2,w_1. The last element of the basis is below, and it has one unknown, which is w_1.

$$\begin{aligned} & 2\,m_z\,n_y\,m_y\,n_z - {m_z}^2{n_y}^2 - {m_y}^2{n_z}^2 - {m_y}^2{n_x}^2 \\ & +2\,m_y\,n_x\,m_x\,n_y - {m_x}^2{n_y}^2 \\ & +({m_x}^2{n_z}^2 - 2\,m_x\,m_z\,n_x\,n_z + {n_x}^2{m_z}^2 + {m_z}^2{n_y}^2 - 2\,m_z\,n_y\,m_y\,n_z \\ & +{m_y}^2{n_z}^2 + {m_y}^2{n_x}^2 - 2\,m_y\,n_x\,m_x\,n_y + {m_x}^2{n_y}^2){w_1}^2 \end{aligned} \tag{A.15}$$

References

[Ackley, 1987] D. H. Ackley, *A connectionist machine for genetic hill climbing*. Kluwer Academic Publishers, 1987.

[Aho *et al.*, 1975] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*. New York: Addison-Wesley, 1975.

[Amid *et al.*, 1988] S. Amid, B. Trethewey, and C. Archibald, "A system for calibrating the wrist mounted laser rangefinder," Tech. Rep. NRCC No. 29799, National Research Council of Canada, Ottawa, Canada, November 1988.

[Ankenbrandt *et al.*, 1990] C. A. Ankenbrandt, B. P. Buckles, and F. E. Petry, "Scene recognition using genetic algorithms with semantic nets," *Pattern Recognition Letters*, vol. 11, pp. 285–293, Apr. 1990.

[Asal *et al.*, 1986] M. Asal, G. Short, T. Preston, R. Simpson, D. Roskell, and K. Guttag, "The texas instruments 34010 graphics system processor," *IEEE Computer Graphics and Applications*, vol. 6, pp. 24–39, Oct. 1986.

[Aubry and Hayward, 1988] S. Aubry and V. Hayward, "Building recursive solid models from sensor data," Tech. Rep. McRCIM-TR-CIM 88-12, Computer Vision and Robotics Laboratory, McGill Research Center for Intelligent Machines, McGill University, Montréal, Canada, April 1988.

[Austin *et al.*, 1991] W. J. Austin, A. M. Wallace, and V. Fraitot, "Parallel algorithms for plane detection using an adaptive Hough transform," *Image and Vision Computing*, vol. 9, pp. 372–384, Dec. 1991.

[Bajaj and Ihm, 1992] C. Bajaj and I. Ihm, "Algebraic surface design with hermite interpolation," *ACM Transactions on Graphics*, vol. 11, pp. 61–91, Jan. 1992.

[Ballard, 1981] D. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[Beaton and Tukey, 1974] A. Beaton and J. Tukey, "The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data," *Technometrics*, vol. 16, pp. 147–185, 1974.

[Ben-Tzvi *et al.*, 1989] D. Ben-Tzvi, A. Naqvi, and M. Sandler, "Efficient parallel implementations of the Hough transform on a distributed memory system," *Image and Vision Computing*, vol. 7, pp. 167–172, Aug. 1989.

[Bergen and Shvaytser, 1991] J. R. Bergen and H. Shvaytser, "A probabilisitc algorithm for computing Hough transforms," *Journal of Algorithms*, vol. 12, pp. 639–656, 1991.

[Besl and Jain, 1985] P. J. Besl and R. C. Jain, "Three dimensional object recognition," *ACM Computing Surveys*, vol. 17, pp. 75–145, Mar. 1985.

[Besl and McKay, 1992] P. Besl and N. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, Feb. 1992.

[Besl *et al.*, 1988] P. J. Besl, J. B. Birch, and L. T. Watson, "Robust window operators," in *Second International Conference on Computer Vision*, (Tampa, Florida), IEEE Computer Society, Dec. 1988.

[Besl, 1988] P. J. Besl, "Geometric modelling and computer vision," *Proceedings of the IEEE*, vol. 76, pp. 936–958, Aug. 1988.

[Besl, 1990] P. J. Besl, "Geometric signal processing," in *Analysys and Interpretation of Range Images* (R. C. Jain and A. K. Jain, eds.), Perception Eng., ch. 3, New York: Springer Verlag, 1990.

[Bhanu *et al.*, 1991] B. Bhanu, S. Lee, and J. Ming, "Self-optimizing image segmentation system using a genetic algorithm," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, (San Diego), pp. 362–369, July 1991.

[Blais *et al.*, 1991] F. Blais, M. Rioux, and S. Maclean, "Intelligent, variable resolution laser scanner for the space vision system," in *Acquisiton, Tracking and Pointing V*, vol. 1482, (Orlando, Florida), SPIE - The International Society for Optical Engineering, 1991.

[Blake and Zisserman, 1987] A. Blake and A. Zisserman, *Visual Reconstruction.* Cambridge: The MIT Press, 1987.

[Bolle and Vemuri, 1991] R. M. Bolle and B. C. Vemuri, "On three dimensional surface reconstruction methods," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, pp. 1–13, Jan. 1991. Nothing.

[Bolles and Fischler, 1981] R. C. Bolles and M. A. Fischler, "A ransac-based approach to model fitting and its application to finding cylinders in range data," in *Seventh International Joint Conference on Artificial Intelligence*, (Vancouver, British Colombia, Canada), pp. 637–643, 1981.

[Bookstein, 1979] F. L. Bookstein, "Fitting conic sections to scattered data," *Computer Vision, Graphics and Image Processing*, vol. 9, pp. 56–71, 1979.

[Boulanger and Godin, 1992] P. Boulanger and G. Godin, "Multiresolution segmentation of range images based on bayesian decision theory," in *Intelligent Robotics and Computer Vison XI: Algorithms, Techniques and Active Vision*, vol. 1825, pp. 16–18, SPIE - The International Society for Optical Engineering, Nov. 1992.

[Boulanger *et al.*, 1990] P. Boulanger, F. Blais, and P. Cohen, "Detection of depth and orientation discontinuities in range images using mathematical morphology," in *10th International Conference on Pattern Recognition*, (Atlantic City, New Jersey), pp. 729–732, June 1990.

[Bramlette and Cusic, 1989] M. F. Bramlette and R. Cusic, "A comparitive evaluation of search methods applied to parametric design of aircraft," in *Third International Conference on Genetic Algorithms*, (George Mason University, Virginia), pp. 213–218, Morgan Kaufman, June 1989.

[Bresenham, 1965] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[Bresenham, 1977] J. E. Bresenham, "A linear algorithm for incremental display of digital arcs," *Communications of the ACM*, pp. 100–106, Feb. 1977.

[Buchberger, 1985] B. Buchberger, "Gröbner bases: an algorithmic method in polynomial ideal theory," in *Recent trends in multidimensional system theory* (N. Bose, ed.), Hingham, Mass.: Reidel, 1985.

[Buchberger, 1989] B. Buchberger, "Applications of Gröbner bases in non-linear computational geometry," in *Geometric Reasoning* (D. Kapur and J. Mundy, eds.), pp. 413–446, MIT Press, 1989.

[Canny, 1987] J. F. Canny, *The complexity of robot motion planning*. Cambridge: The MIT Press, 1987.

[Chandler, 1988] R. E. Chandler, "A tracking algorithm for implictely defined curves," *IEEE Computer Graphcs and Applications*, pp. 83–89, Mar. 1988.

[Char *et al.*, 1988] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt, *Maple Reference Manual: Fifth Edition*. Waterloo, Ontario, Canada: Symbolic Computation Group University of Waterloo, 1988.

[Corana *et al.*, 1987] M. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the simulated annealing algorithm," *ACM Transactions on Mathematical Software*, vol. 13, pp. 262–280, Sep. 1987.

[Darrell *et al.*, 1990] T. Darrell, S. Sclaroff, and A. Pentland, "Segmentation by minimal description," in *Third International Conference on Computer Vision*, (Osaka, Japan), pp. 112–116, Dec. 1990.

[Davies, 1987] E. R. Davies, "A modified Hough scheme for general circle location," *Pattern Recognition Letters*, vol. 7, pp. 37–43, Jan. 1987.

[Dillencourt *et al.*, 1992] M. Dillencourt, D. Mount, and N. Netanyahu, "A randomized algorithm for slope detection," *International Journal of Computational Geometry and Applications*, vol. 2, no. 1, pp. 1–27, 1992.

[Duda and Hart, 1971] R. O. Duda and P. E. Hart, "The use of the Hough transform to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, pp. 11–15, 1971.

[Duncan, 1990] R. Duncan, "A survey of parallel computer architechtures," *IEEE Computer*, vol. 23, pp. 5–16, Feb. 1990.

[Fan *et al.*, 1987] T. J. Fan, G. Medioni, and R. Nevatia, "Segmented descriptions of 3-d surfaces," *IEEE Journal of Robotics and Automation*, vol. 3, pp. 527–538, Dec. 1987.

[Farin, 1993] G. Farin, *Curves and surfaces for computer-aided geometric design*. Boston: Academic Press, 1993.

[Faux and Pratt, 1979] I. D. Faux and M. J. Pratt, *Computational geometry for design and manufacture*. Ellis Horwood, 1979.

[Fischler and Bolles, 1981] M. A. Fischler and R. C. Bolles, "Random sample consensus," *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.

[Fischler and Firschein, 1987] M. A. Fischler and O. Firschein, "Parallel guessing: a strategy for high-speed computation," *Pattern Recognition*, vol. 20, no. 2, pp. 257–263, 1987.

[Fisher and Highnam, 1989] A. L. Fisher and P. T. Highnam, "Computing the Hough transform on a scan line array processor," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 262–265, Mar. 1989.

[Foley and Dam, 1982] J. D. Foley and A. V. Dam, *Fundamentals of Interactive Computer Graphics*. Reading, Mass.: Addison-Wesley, 1982.

[Forstner, 1987] W. Forstner, "Reliability analysis of parameter estimation in linear models with applications to mensuration problems in computer vision," *Computer Vision, Graphics and Image Processing*, vol. 40, pp. 273–310, 1987.

[Forstner, 1989] W. Forstner, "Robust statistics methods for computer vision," in *Tutorial, Conference on Computer Vision and Pattern Recognition*, (San Diego, California), 1989.

[Gatermann, 1990] K. Gatermann, "Symbolic solution of polynomial equations with symmetry," in *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, (Tokyo, Japan), pp. 112–119, Addison Wesley, 1990.

[Geman and Geman, 84] S. Geman and D. Geman, "Stochastic relaxation, gibbs distribution, and bayesian restoration of images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 84.

[Gentleman *et al.*, 1987] W. M. Gentleman, M. Wein, and D. Green, "Harmony, an operating system for embedded industrial multiprocessor applications," in *Proc. of IEEE Montech — COMPINT '87*, (Montreal, Quebec, Canada), Nov. 1987.

[Gentleman, 1965] W. M. Gentleman, *Robust estimation of multivariate location by minimizing pth power deviatons.* PhD thesis, Princeton University, 1965.

[Giraudon, 1987] G. Giraudon, "A real time parallel edge following in a single pass," in *Proceedings of the IEEE Workshop on Computer Vision*, (Miami Beach, Florida), pp. 228–230, November. 1987.

[Godin and Levine, 1989] G. D. Godin and M. D. Levine, "Structured edge maps of curved objects in a range image," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Diego, California), pp. 276–281, 1989.

[Goldberg, 1988] D. Goldberg, *Genetic algorithms in search, optimization and machine learning.* Reading, Mass.: Addison-Wesley, 1988.

[Grimson and Huttenlocher, 1990] E. L. Grimson and D. P. Huttenlocher, "On the sensitivity of the Hough transform for object recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, pp. 255–274, Mar. 1990.

[Hampel *et al.*, 1986] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel, *Robust statistics: the approach based on influence functions.* New York: Wiley, 1986.

[Hegron, 1988] G. Hegron, *Image synthesis.* Cambridge, Mass.: MIT Press, 1988.

[Hill and Taylor, 1992] A. Hill and C. J. Taylor, "Model-based image interpretation using genetic algorithms," *Image and Vision Computing*, vol. 10, pp. 295–300, June 1992.

[Hillis, 1985] W. D. Hillis, *The connection machine.* ACM Distinguished Dissertation, Cambridge, Massachusetts: The MIT Press, 1985.

[Hoffman and Jain, 1987] R. Hoffman and A. K. Jain, "Segmentation and classification of range images," *IEEE Transactions On Pattern Analysis and Machine Intelligence*, vol. 9, pp. 608–620, Sep. 1987.

[Hoffman, 1989] C. M. Hoffman, *Geometric and solid modelling.* San Mateo, California: Morgan Kaufman, 1989.

[Holland, 1975] J. H. Holland, *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[Holland, 1992] J. H. Holland, "Genetic algorithms," *Scientific American*, pp. 66–72, July 1992.

[Hook and McAree, 1990] D. G. Hook and P. R. McAree, "Using sturm sequences to bracket real roots of polynomial equations," in *Graphics gems* (A. Glassner, ed.), Academic Press, 1990.

[Hoppe *et al.*, 1992] H. Hoppe, T. DeRose, T. Duchamp, and J. McDonald, "Surface reconstuction from unorganized points," in *ACM Computer Graphics*, vol. 26, pp. 71–78, July 1992.

[Huang, 1989] C. L. Huang, "Elliptical feature extraction via an improved Hough transform," *Pattern Recognition Letters*, vol. 10, pp. 93–100, 1989.

[Huber, 1981] P. Huber, *Robust statistics.* New York: Wiley, 1981.

[Illingworth and Kittler, 1988] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Vision, Graphics and Image Processing*, vol. 44, pp. 87–116, 1988.

[Jolion *et al.*, 1991] J. M. Jolion, P. Meer, and S. Bataouche, "Robust clustering with applications in computer vision," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, pp. 791–802, Aug. 1991.

[Kierkeggaard, 1992] P. Kierkeggaard, "A method for detection of circular arcs based on the Hough transform," *Machine Vision and Applications*, vol. 5, no. 249-263, 1992.

[Kim *et al.*, 1989] D. Y. Kim, J. J. Kim, P. Meer, D. Mintz, and A. Rosenfeld, "Robust computer vision: A least median of squares approach," in *DARPA Image Understanding Workshop*, pp. 1117–1134, May 1989.

[Kiryati *et al.*, 1991] N. Kiryati, Y. Eldar, and A. M. Bruckstein, "A probabilistic Hough transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303–316, 1991.

[Kriegman and Ponce, 1990] D. J. Kriegman and J. Ponce, "On recognizing and positioning curved 3-d objects from image contours," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, pp. 1127–1137, Dec. 1990.

[Kumar and Krishnan, 1989] V. K. Kumar and V. Krishnan, "Efficient parallel algorithms for image template matching on hypercube simd machines," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 665–669, June 1989.

[Leclerc, 1989] Y. G. Leclerc, "Constructing simple stable descriptions for image partitioning," *International Journal of Computer Vision*, vol. 3, pp. 73–102, May 1989.

[Levine, 1985] M. D. Levine, *Vision in man and machine.* New York: McGraw-Hill, 1985.

[Little *et al.*, 1989] J. L. Little, G. E. Belloch, and T. A. Cass, "Algorithmic techniques for computer vision on a fine-grained parallel machine," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 244–257, Mar. 1989.

[Macaulay, 1916] F. Macaulay, *The algebraic theory of modular systems.* Cambridge University Press, 1916.

[Manderick and Speissens, 1989] B. Manderick and P. Speissens, "Fine grained parallel genetic algorithms," in *Proceedings of Third International Conference on Genetic Algorithms*, (George Mason University), pp. 428–433, June 1989.

[Meer *et al.*, 1990] P. Meer, D. Mintz, and A. Rosenfeld, "Least median of squares based robust analysys of image structure," Tech. Rep. CAR-TR-490, Center for Automation Research, University of Maryland, College Park, Maryland, Mar. 1990.

[Meer *et al.*, 1991] P. Meer, D. Mintz, A. Rosenfeld, and D. Y. Kim, "Robust regression methods in computer vision: a review," *International Journal of Computer Vision*, vol. 6, no. 1, pp. 59–70, 1991.

[Meer, 1991] P. Meer, "Robust high breakdown estimation and consensus," in *Vision Interface 91*, (Calgary, Alberta), pp. 112–116, June 1991.

[Meygret *et al.*, 1992] A. Meygret, M. D. Levine, and G. Roth, "Robust primitive extraction in a range image," in *11th International Conference on Pattern Recognition: Image, Speech and Signal Analysis*, (The Hague, Netherlands), pp. 193–196, Aug. 1992.

[Mintz, 1991] D. Mintz, "Robustness by consensus," Tech. Rep. CAR-TR-576, Center for Automation Research, University of Maryland, College Park, Maryland, Aug. 1991.

[Morgan, 1987] A. Morgan, *Solving polynomial systems using continuation for science and engineering.* Prentice Hall, Englewoord Cliffs, 1987.

[Nalwa and Pauchon, 1987] V. S. Nalwa and E. Pauchon, "Edge aggregation and description," *Computer Vision, Graphics and Image Processing*, vol. 40, pp. 79–94, 1987.

[Netanyahu, 1992] N. Netanyahu, "Computationally efficient algorithms for robust estimation," Computer Science Technical Report Series CS-TR-2898, University of Maryland, 1992.

[Oja and Xu, 1990] P. K. E. Oja and L. Xu, "A new curve detection method: randomized Hough transform (rht)," *Pattern Recognition Letters*, vol. 11, pp. 331–338, May 1990.

[Pednault, 1989] E. P. Pednault, "Some experiments in applying inductive inference principles to surface reconstruction," in *Eleventh international conference on artificial intelligence*, (Detroit, Michigan, USA), pp. 1603–1610, Aug. 1989.

[Piegl, 1991] L. Piegl, "On nurbs: a survey," *IEEE Computer Graphics and Applications*, pp. 55–91, Jan. 1991.

[Ponce *et al.*, 1991] J. Ponce, A. Hoggs, and D. J. Kriegman, "On using cad models to compute the pose of curved 3d objects," in *Workshop on Directions in Automated CAD Vision*, (Maui, Hawaii), pp. 136–145, IEEE Computer Society, 1991.

[Pratt, 1987] V. Pratt, "Direct least squares fitting of algebraic surfaces," *Computer Graphics*, vol. 21, pp. 145–152, July 1987.

[Press and Flannery, 1988] W. H. Press and B. P. Flannery, *Numerical recipes in C*. Cambridge university press, 1988.

[Princen *et al.*, 1989] J. Princen, J. Illingworth, and J. Kittler, "Templates and the Hough transform," in *NATO ASI on Active Perception and Robot Vision*, (Maratea, Italy), July 1989.

[Renegar, 1987] J. Renegar, "On the efficiency of newton's method in approximating all zeros of a system of a complex polynomials," *Mathematics of Operations Research*, vol. 12, pp. 121–148, Feb. 1987.

[Rioux and Blais, 1986] M. Rioux and F. Blais, "Compact three dimensional camera for robotic applications," *Journal of the Optical Society of America*, vol. 3, pp. 1518–1520, 1986.

[Rioux, 1985] M. Rioux, "Laser rangefinders based on synchronized scanning," *Applied Optics*, vol. 23, pp. 3837–3844, 1985.

[Rosenfeld *et al.*, 1988] A. Rosenfeld, J. Ornelas, and Y. Hung, "Hough transform algorithms for mesh-connected simd parallel processors," *Computer Vision, Graphics and Image Processing*, vol. 41, pp. 293–305, Mar. 1988.

[Roth and Levine, 1990a] G. Roth and M. D. Levine, "Random sampling for primitive extraction," in *International Workshop on Robust Computer Vision*, (Seattle, Washington), Oct. 1990.

[Roth and Levine, 1990b] G. Roth and M. D. Levine, "Segmentation of geometric signals using robust fitting," in *10th International Conference on Pattern Recognition: Pattern Recognition Systems and Applications*, (Atlantic City, New Jersey), pp. 826–831, June 1990.

[Roth and Levine, 1991a] G. Roth and M. D. Levine, "A genetic algorithm for primitive extraction," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, (San Diego), pp. 487–494, July 1991.

[Roth and Levine, 1991b] G. Roth and M. D. Levine, "Random sampling for pose determination and refinement," in *Vision Interface 91*, (Calgary, Alberta), pp. 104–111, June 1991.

[Roth and Levine, 1991c] G. Roth and M. D. Levine, "Robustness in primitive extraction and correspondence computation," in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611, pp. 542–554, SPIE - The International Society for Optical Engineering, 1991.

[Roth and Levine, 1992a] G. Roth and M. D. Levine, "Extracting geometric primitives," Tech. Rep. McRCIM-TR-CIM 92-13, Computer Vision and Robotics Laboratory McGill Research Center for Intelligent Machines, McGill University, Montréal, Canada, Oct. 1992.

[Roth and Levine, 1992b] G. Roth and M. D. Levine, "Geometric primitive extraction using a genetic algorithm," Tech. Rep. McRCIM-TR-CIM 92-14, Computer Vision and Robotics Laboratory McGill Research Center for Intelligent Machines, McGill University, Montréal, Canada, Oct. 1992.

[Roth and Levine, 1992c] G. Roth and M. D. Levine, "Geometric primitive extraction using a genetic algorithm," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 640–643, Champaign, Illinois: IEEE Computer Society Press, June 1992.

[Roth and Levine, 1992d] G. Roth and M. D. Levine, "Minimal subset random sampling for pose determination and refinement," in *Advances in Machine Vision: Strategies and Applications* (C. Archibald and E. Petriu, eds.), vol. 32 of *Computer Science*, pp. 1–21, Singapore, Singapore: World Scientific, 1992.

[Roth and Levine, In Press] G. Roth and M. D. Levine, "Extracting geometric primitives," *Computer Vision, Graphics and Image Processing: Image Understanding*, (In Press).

[Rousseeuw and Leroy, 1987] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection.* Wiley, 1987.

[Rousseeuw, 1984] P. J. Rousseeuw, "Least median of squares regression," *Journal of American Statistical Association*, vol. 79, pp. 871–880, Dec. 1984.

[Samet, 1984] H. Samet, "The quadtree and related hierarchical data structure," *ACM Computing Surveys*, vol. 16, pp. 187–260, 1984.

[Sampson, 1982] P. D. Sampson, "Fitting conic sections to "very scattered" data," *Computer Vision, Graphics and Image Processing*, vol. 18, pp. 97–108, Jan. 1982.

[Sarkar and Menq, 1992] B. Sarkar and C. Menq, "Parameter optimizatin in approximating curves and surfaces to measurement data," *Computer Aided Geometric Design*, vol. 8, pp. 267–290, 1992.

[Sederberg and Anderson, 1984] T. W. Sederberg and D. C. Anderson, "Implicit representation of parametric curves and surfaces," *Computer Vision, Graphics and Image Processing*, vol. 28, pp. 72–84, 1984.

[Sederberg and Anderson, 1985] T. Sederberg and D. Anderson, "Steiner surface patches," *IEEE Computer Graphics and Applications*, pp. 23–36, May 1985.

[Sederberg, 1985] T. Sederberg, "Piecewise algebraic surface patches," *Computer Aided Geometric Design*, vol. 2, no. 53-59, 1985.

[Sklansky, 1991] J. Sklansky, "Machine vision needs a robust technology," *Machine Vision and Applications*, vol. 4, pp. 59–87, 1991.

[Spears and DeJong, 1991] W. M. Spears and K. A. DeJong, "On the virtues of parameterized uniform crossover," in *Proceedings of the Third International Conference on Genetic Algorithms* (M. Kaufman, ed.), (San Diego, California), pp. 230–236, 1991.

[Stewart, 1991] C. V. Stewart, "Robust surface reconstruction based on local estimation and refinement: 1d-results," in *Sensor Fusion IV*, vol. 1611, (Boston, Mass.), SPIE, Nov. 1991.

[Stockman and Agrawala, 1977] G. C. Stockman and A. K. Agrawala, "Equivalence of Hough transform to template matching," *Communications of the ACM*, vol. 20, pp. 820–822, 1977.

[Syswerda, 1989] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the third international conference on genetic algorithms*, (George Mason University, Fairfax, Virginia), pp. 2 - 9, July 1989.

[Tanese, 1989] R. Tanese, "Distributed genetic algorithms," in *Proceedings of Third International Conference on Genetic Algorithms*, (George Mason University), pp. 434–439, 1989.

[Taubin and Cooper, 1990] G. Taubin and D. B. Cooper, "Recognition and positioning of piecewise algebraic surfaces," in *Image Understanding Workshop*, pp. 508–514, DARPA, Sep. 1990.

[Taubin, 1988] G. Taubin, "Nonplanar curve and surface estimation in 3-space," in *IEEE International Conference on Robotics and Automation*, pp. 644–645, Apr. 1988.

[Taubin, 1991] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves with applications to edge and range segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, pp. 1115–1138, Nov. 1991.

[Tirunmalai and Schunk, 1989] A. Tirunmalai and B. G. Schunk, "Robust surface approximation using least median of squares approach," Tech. Rep. CSE-TR-13-89, University of Michigan, Computer Science and Electrical Engineering, Ann Arbour, Michigan, 1989.

[VanAken, 1984] J. R. VanAken, "An efficient ellipse drawing algorithm," *IEEE Computer Graphics and Automation*, vol. 4, pp. 26–34, Sep. 1984.

[Vishkin, 1983] U. Vishkin, "Synchronous parallel computation - a survey," Tech. Rep. 69, Computer Science Department, New York Univ., New York, New York, 1983.

[Wallis, 1990] B. Wallis, "Rendering fat lines on a raster grid," in *Graphics Gems* (A. Glassner, ed.), Addison-Wesley, 1990.

[Walpole and Myers, 1989] R. E. Walpole and R. H. Myers, *Probability and statistics for scientists and engineers.* New York: Macmillan Publishing, 1989.

[Wilkinson, 1959] J. H. Wilkinson, "The evaluation of the zeroes of ill-conditioned polynomials," *Num. Math.*, vol. 1, pp. 150–180, 1959.

[Yip *et al.*, 1992] R. Yip, P. Tam, and D. Leung, "Modifications of the Hough transform for circles and ellipse detection using a 2-dimensional array," *Pattern Recognition*, vol. 25, no. 9, pp. 1007–1022, 1992.

[Yokoya and Levine, 1989] N. Yokoya and M. D. Levine, "Range image segmentation based on differential geometry: a hybrid approach," *IEEE Transactions On Pattern Analysis and Machine Intelligence*, vol. 11, pp. 745–758, July 1989.

[Yuen *et al.*, 1989] H. Yuen, J. Illingworth, and J. Kittler, "Detecting partially occluded ellipses using the Hough transform," *Image and Vision Computing*, vol. 7, Feb. 1989.

[Zhuang *et al.*, 1992] X. Zhuang, T. Wang, and P. Zhang, "A highly robust estimator through partially likelihood function modelling and its application in comptuter vision," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, pp. 19–35, Jan. 1992.

Author Index