Reconstruction and Identification of Features in Planar and 3D Objects

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Michael J. Donahue

*****

The Ohio State University

1992

Dissertation Committee

Stanislav Rokhlin

Laslo Adler

Carolyn Merry

Roni Yagel

Adviser's Approval

Adviser
Department of Welding Engineering

To My Mother,

Ruth M. Donahue

Acknowledgements

Let me express my genuine appreciation to my thesis advisor, Professor Stanislav Rokhlin. Without his guidance and persistence this document could not have been brought to fruition. Also my thanks to my dissertation committee, Professors Laslo Adler, Carolyn Merry, and Roni Yagel; I sincerely appreciate their time, effort, and enthusiasm.

I am also grateful to my fellow graduate students. I cannot mention them all, but allow me to make special acknowledgements to Wei Huang, Ligou Wang, and Dan Applegate. The camaraderie shared by graduate students is an important aspect of the graduate school experience that should not be discounted. My thanks also, of course, to my family and friends, for their steadfast support and devotion throughout the many years of my education. In particular, let me express my deepest appreciation to my mother, who more than anyone else has provided support when I needed it most.

Finally, let me recognize the Edison Welding Institute, Kodak, and the American Society for Nondestructive Testing for providing financial support for this research.

Vita

May 31, 1962	born—Columbus, Ohio
1981–1983	Undergraduate Research Assistant, University of Dayton Research Institute, Dayton, Ohio
1984	B.S. Electrical Engineering, The Ohio State University, Columbus, Ohio
1984–1987	Graduate Teaching Associate, Department of Mathematics, The Ohio State University
1985	M.S. Mathematics, The Ohio State University
1991	Ph.D. Mathematics, The Ohio State University
1987–Present	Graduate Research Associate, Department of Welding Engineering, The Ohio State University

Publications

1. M. J. Donahue, A. P. Sprague and S. I. Rokhlin, "Point Matching Method for Flaw Detection in Printed Circuit Boards," in *Review of Progress in Quantitative NDE*, D. O. Thompson and D. E. Chimenti eds., **8B**, 1233–1240 (Plenum Press, New York, 1989).

2. A. P. Sprague, M. J. Donahue and S. I. Rokhlin, "A Method for Automatic Inspection of Printed Circuit Boards," *Computer Vision, Graphics and Image Processing: Image Understanding*, **54**, 401–415 (1991).

Fields of Study

Major Field: Welding Engineering

Table of Contents

List of Tables

List of Figures

FIGURE **PAGE**

Introduction and objectives

With the increasing complexity of manufacturing techniques, the nondestructive evaluation of products becomes an important component in maintaining product quality and increasing structural reliability. One such method, developed originally in medicine and more recently in industry, is computerized tomography. Here we study the problem of tomographic reconstruction from incomplete data, which is an important problem in industrial tomography not present in the medical setting. Modern sophisticated nondestructive evaluation techniques, tomography and others, generate large amounts of data. It is therefore important to develop computerized methods to analyze this data. One such method that will be investigated in this work is computer-automated feature extraction from images.

Radiography is commonly used for flaw detection in a variety of objects. However, radiographs can be difficult to interpret, especially if the object has a complicated geometry or internal structure. Tomographic reconstructions allow one to see into an object by producing a cross-sectional view of the object density. Such reconstructions are easily interpreted and compared to part specifications.

Reconstructions of this type produce cross-sectional images of incredible accuracy, often on the order of one part in a thousand (see [2]). However, conventional tomographic reconstructions, which were developed in the medical field, require radiographs from all directions surrounding the object. In industrial applications body geometry, surrounding structures, or other restrictions can make such complete data collection impossible. Reconstruction from a restricted data set of this type is known as the limited angle observation problem. On the other hand, industrial tomography, as compared to as medical tomography, does not have restrictions on radiation dosage (due to the absence of biological constraints) and may have a priori information available. As an example of the latter, in industrial applications one may test for voids in a material where the surrounding material is known to be homogeneous. Alternatively, one could test manufactured parts using the part specification for comparison. In both cases a significant amount of information is available before the part undergoes any radiation exposure. (More detailed background information on tomography is provided in Chapter I.)

Volume reconstructions from computerized tomography or multidirectional radiography produce enormous amounts of data, which raises the additional problem of automated defect recognition. Similar problems arise in real-time automated optical and radiographic inspection systems, where automated inspection is required in order to provide both reliability and speed. To deal with the vast amount of data contained in a raw image, it is useful to reduce the image to a smaller set of

important features, and then do subsequent processing on this reduced set.

Feature extraction and identification is a general technique used in many areas of computer vision. In radiographs, for example, material inhomogeneities and discontinuities should be detected. Similarly, trace endings from printed circuit board images should be extracted for optical detection of defects. Any new or missing endings (as compared to the original circuit specification) correspond to trace break or join defects on the board. Fingerprint identification is done in an analogous fashion by matching fingerprint ridge endings and bifurcations. In all examples of this type, if the features can be reliably extracted, then image recognition can be done in a robust manner.

Objectives

This dissertation research has two major objectives. First, to develop new methods in the limited angle problem using a priori information available in industrial settings. Second, to develop methods for automated feature extraction and identification from general images.

These objectives will be met as follows. First a study of 3D cone beam tomography will be done, including algorithmic implementation and a theoretical understanding of the reconstruction limitations arising from beam scanning geometry. Then the limited angle problem will be considered, with possible solutions involving analytic continuation and iterative processing using a priori information.

As an aid to these studies I will implement a 3D radiograph simulation program. As a first step towards feature extraction and identification, I will present a method for the extraction of image topological information from general images and show the use of this information for feature detection in a variety of images. A detailed study will then be made of feature extraction and image identification, with applications including the use of the above topological information to radiographic images, printed circuit boards and fingerprints.

This dissertation is structured as follows. Part I is devoted to tomographic reconstruction of objects. First is some historical and theoretical background material and a review of reconstruction techniques currently in use. Following this I present an elegant radiograph simulation technique for 3 dimensional bodies, based on the superposition of analytically calculated radiographs for several base element types such as spheres, ellipsoids, and rectangular solids. This is useful independently for the study of radiographs of complex objects, but was also needed to generate simulated radiographs for the tomographic reconstruction algorithms developed as part of this work. These simulations were used, for example, in the development of the 3 dimensional cone beam reconstruction algorithm presented in Chapter III. This algorithm is an implementation of an idea of Feldkamp [3].

Chapter IV contains new results in tomographic reconstructions from limited angle data. Section 4.1 details the development and results of the implementation of an analytic continuation technique suggested by Palamodov and Denisjuk

[4]. This technique uses the known data to extrapolate data in the missing directions. To my knowledge this is the first actual implementation of this method. In comparison, Section 4.3 is more pedestrian. This section concentrates on the iterative (algebraic) reconstruction technique and the use of a priori data. Several new ideas are introduced to the reconstruction method, but the main thrust of the section is towards a quantitative evaluation of the effects of projection order and two different types of a priori information.

The second part of this dissertation focuses on feature recognition in planar images. In general the material in this part could be developed for the detection of flaws (or other features) in radiographs or tomographic reconstructions, but the funding for our research was such that instead the examples presented relate to printed circuit board inspection and fingerprint identification.

Chapter VII presents some general techniques for image analysis on grey level images, including examples spanning a variety of images. The succeeding chapter presents feature extraction on printed circuit boards, where several new algorithms are suggested. Chapter IX provides extensive detail on an original fingerprint feature extraction system. This system, developed on an IBM PC platform, compares favorably in terms of extraction quality with commercial systems. Finally, Chapter X details methods for feature classification through matching. In particular, Section 10.1 describes a method for determining which printed circuit board features (extracted using the methods of Chapter VIII) correspond to circuit defects,

and Section 10.2 develops a complex method for latent fingerprint identification based on features extracted using the methods of Chapter IX.

Part I

Tomographic reconstruction of planar and 3D objects

CHAPTER I

Background to Part I

1.1 History

Tomography is the reconstruction of a body from its projections. In practice, a body of unknown varying densities is exposed to X-ray radiation from one direction and a radiograph (or rather a digital representation of one) is produced. Then the body (or alternately the X-ray source) is rotated, and a radiograph is taken from a different direction. This process is repeated for many angles completely surrounding the body. The obtained radiographic data, referred henceforth as "projection" data, is then mathematically inverted to recover the original (unknown) body densities.

The history of tomography traces back to 1917, when J. Radon published a solution to the inversion problem for what is now known as the Radon transform [5]. (In 2 dimensions the Radon transform is equivalent to the X-ray transform used in X-ray transmission tomography.) Although Bracewell and Riddle [6, 7] studied the Radon inversion problem in radio astronomy from the 1950's, the field

was relatively dormant until the early 1960's, when A. Cormack constructed a parallel beam X-ray scanner which used an algebraic reconstruction technique for radiological applications [8, 9]. Great interest in the medical community led to the development of the first commercial **CAT** (Computer Axial Tomography) scanner in 1972 by Hounsfield [10]. In 1979 Cormack and Hounsfield received the Nobel prize in medicine for their (independent) accomplishments.

Rapid advances in computerized tomography through the 1970's was due mainly to developments in reconstruction algorithms. The early devices used an algebraic reconstruction technique (ART), which has been replaced in modern implementations with a more computationally efficient technique called filtered backprojection (FBP). The filtered backprojection method was used as early as 1967 in radio astronomy [7], but was introduced to the medical literature by Ramachandran and Lakshminarayanan [11] and Shepp and Logan [12].

Although the filtered backprojection algorithm achieves excellent results in medical tomography, it requires a large number of projections uniformly spaced around the body. The algebraic methods are generally more easily adapted to limited data type problems. This class of problems commonly occurs in industrial tomography, and is discussed in more detail in Chapter IV, as well as in general tomographic reconstruction references [13, 14, 2]. The particular problem of having only a restricted range of projection angles available is known as the limited angle problem. Significant contributors to this problem since the late 1970's include

Lewitt [15], Louis [16, 17], Tuy [18], K.T. Smith [19], and M.E. Davison [20, 21].

Another area of development in the 1970's was scanning beam geometry. Early tomographic scanners and reconstruction methods used what is called "parallel beam" geometry. Parallel scanning requires a collimated X-ray source and detector, which are moved in parallel across the body being examined, producing a radiograph strip for one projection direction. This parallel scan is required for each projection direction. As this technique is time consuming and mechanically complicated, later scanners make use of the "fan beam" geometry, in which an X-ray point source (or approximation thereto) irradiates the entire object, producing a radiographic projection from that direction at once. The extension of this fan-beam geometry to 3 dimensions is known as the "cone beam" geometry, and is discussed in detail in Chapter III.

1.2 Theory

Let $\alpha(x)$ denote the X-ray linear attenuation coefficient for a given body at point x. (Although the linear attenuation is not the same as the object density, the two are related, and henceforth the distinction will be disregarded.) Then X-rays passing through the body at point x with intensity I suffer a relative intensity loss of

$$\Delta I / I = \alpha(x) \Delta x \tag{1.1}$$

over a small distance Δx. Therefore if I_0 is the original intensity of the X-ray beam before entering the body, and L denotes the line of travel of the beam, then the intensity I_1 upon exiting the body is given by

$$I_1 = I_0 \exp\left\{-\int_L \alpha(x)\, dx\right\}. \tag{1.2}$$

The physics of X-ray radiation and absorption are discussed in many sources. For example, see [22], pages 71–84.

To simplify discussion let us restrict our attention to the 2 dimensional case. Analogous remarks apply to the higher dimensional situations as well. Let $L_\theta(t)$ denote the line in the xy-plane which makes angle θ with the y-axis at distance t from the origin. Then the projection $P_\theta(t)$ in the direction θ at offset t is given by

$$P_\theta(t) = \int_{L_\theta(t)} \alpha(x(s), y(s))\, ds, \tag{1.3}$$

where s denotes a parameterization of the line L.

According to the Fourier Slice Theorem (see for example [14, page 11]), the 1 dimensional Fourier transform of $P_\theta(t)$ in t is the same as the 2 dimensional Fourier transform of the original function $\alpha(x, y)$, only in polar coordinates. To be precise,

$$\mathcal{F}_{x\to u, y\to v}(\alpha)(\sigma\cos\theta, \sigma\sin\theta) = \mathcal{F}_{t\to\sigma}(P_\theta)(\sigma). \tag{1.4}$$

(Here $\mathcal{F}$ denotes the Fourier transform.) Therefore, by taking one dimensional Fourier transforms of the (known) projection data, one can recover the 2 dimensional Fourier transform of the original (unknown) attenuation function $\alpha(x, y)$.

Since the Fourier transform is invertible, it is possible to recover $\alpha(x, y)$ by taking the inverse 2 dimensional Fourier transform of the transformed projection data. This procedure is known as the Fourier inversion technique.

The chief difficulty with the Fourier inversion technique is that the typical numerical method for the Fourier inversion (the inverse fast Fourier transform, IFFT) requires a square grid, whereas the data obtained after taking the Fourier transform of the projection data lies on a radial grid. Moreover, the necessary interpolation is difficult to perform. However, the inversion steps can be combined and rewritten using polar coordinates. The resulting technique is the filtered backprojection method (FBP), which is extremely fast and accurate. It is upon this method that the cone beam reconstruction algorithm of Feldkamp [3] presented in Chapter III is built. It is also the method used almost universally in medical CT machines.

The speed of the filtered backprojection method is based upon the computational refinements of the fast Fourier transform (FFT). In particular, the filtered backprojection method requires a large number of projections that are spaced evenly around the irradiated body. It is common in industrial applications that some angles are inaccessible, whether due to part geometry (for example, the long axes of a T-joint) or a surrounding structure. Although it is possible to modify the FBP algorithm for such situations, the modifications are nontrivial and the results are disappointing. (See in particular [20].) It is also difficult to incorporate many types of a priori data into the FBP algorithm.

Algebraic reconstruction methods (ART, SIRT, SART—see [2]) are not based on the Fourier transform, but rather on an iterative scheme for solving linear equations developed by Kaczmarz [23]. For this reason these methods are also known as iterative reconstruction methods. They do not in general require modification to handle irregularly spaced projection data and are more flexible with respect to the introduction of a priori data than the Fourier reconstruction methods (including in particular the FBP). Therefore, modern algebraic reconstruction methods, direct descendants of the reconstruction algorithms used in the first CT scanners, are of continuing interest to researchers.

CHAPTER II

Simulation of radiographic projections of 3D objects

2.1 Problem description

To study tomographic reconstructions, it is necessary to understand the formation of the input data used in such reconstructions—the projection data. This chapter details a radiograph simulation package developed both to promote an understanding of projection data and to generate "ideal" (i.e., noise-free) data for reconstruction algorithm testing.

Given a direction and a 3 dimensional object with known (X-ray) attenuations, how can one determine the radiographic projection of the object in the given direction? The radiograph is determined by the integral of the body attenuations along the X-ray paths, so the most general way is to trace each X-ray path through the object, sample the object attenuations along the ray, and use this information to perform a numerical integration. This approach is very flexible, but also runs high computational costs. A faster method is to restrict the allowed objects to be

those that can be constructed by the superposition of objects from some simple family of base element types. If the radiographs of the base elements can be calculated quickly in an analytic fashion, then the principle of superposition can be used to quickly construct the radiograph of any object built from these elements. It is this latter approach that is developed here. The reader may refer to [24] as a general reference on ray tracing methods.

2.2 Theory

Refer to Fig. 2.1, which is an illustration of a T-joint with a sample X-ray path. This path is a straight line, which we shall refer to as L. The intensity of the X ray is attenuated as it traverses through the material. The attenuation depends on the **linear attenuation coefficient**, μ, of the material, which is a function primarily of material density. Formally, the resulting intensity I of an X ray with initial intensity I_0 passing along line L through an object with linear attenuation coefficient $\mu(x)$ is given by

$$I = I_0 \exp\left\{-\int_L \mu(x)\, dx\right\}. \tag{2.1}$$

If the object is homogeneous (i.e., $\mu(x)$ is constant inside the body, 0 outside), then this reduces to

$$I = I_0 \exp\left\{-\mu \| L \cap \text{Body} \|\right\}, \tag{2.2}$$

where $\| L \cap \text{Body} \|$ is the length of the intersection of the line L with the body. We will call $\mu \| L \cap \text{Body} \|$ the **linear attenuation due to the body**.

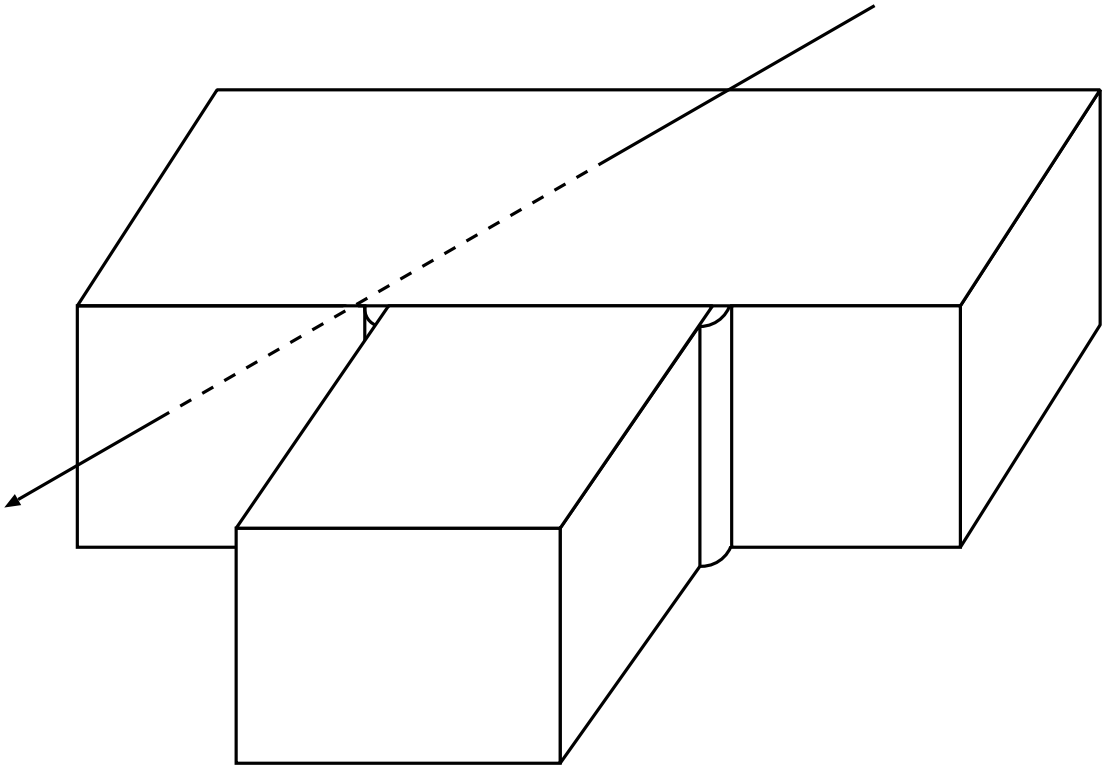


Figure 2.1: T-joint with trace of an X-ray path.

So assume that we have an object for which we know the linear attenuation (as a function in 3-space), and suppose we would like to simulate a radiograph of this object from a given direction. As in any computerized approach, we must first change the problem into a discrete one. In particular, the simulated radiograph will consist of a finite (though large) number of picture elements (called **pixels**) arranged in a rectangular grid, each corresponding to one photon path. (Alternately, each pixel can be associated to a narrow pencil beam of X rays.) The simulation software must calculate the attenuation (via Eq. 2.1 or 2.2) along the X-ray path for each pixel in the display.

One straightforward technique for these calculations is **ray tracing**. In this method the integral in Eq. 2.1 is directly approximated as the Riemann sum

$$I \approx I_0 \exp\left\{-\sum_k \mu(x_k)\,\Delta x\right\}, \tag{2.3}$$

where the points x_k are spaced Δx apart along the line L. To get reasonable accuracy from this method, the step size Δx must be made small. When one takes into account the fact that these calculations must be made for each pixel, and the number of pixels is large, one sees that the computational requirements for this approach are monumental. For example, the examples in this report are based on a 200×200 pixel grid. If the summation above is taken over 100 steps (an absolute minimum), then the linear attenuation function must be evaluated at 4 million different points. In addition there is the overhead of actually calculating

the coordinates of each point.

The ray tracing method has high computational requirements, although it is a very general method that can be applied to any object geometry. Suppose, however, that the object is composed of several homogeneous pieces. Then the linear attenuation can be calculated as in Eq. 2.2 for each piece, and then the resulting linear attenuation for the entire object is given via the principle of superposition. In particular, if the object is composed of n homogeneous pieces with linear attenuations coefficients μ_1, μ_2, ..., μ_n, and path-body intersection lengths of l_1, l_2, ..., l_n (for a given X-ray path), then the attenuation for the object as a whole (for the given X-ray path) is given by

$$I/I_0 = \exp(-\mu_1 l_1 - \mu_2 l_2 - \ldots - \mu_n l_n), \tag{2.4}$$

where $\mu_k l_k$ $(k = 1, 2, \ldots, n)$ is the linear attenuation for piece k. In this method only the intersection lengths l_k need to be calculated for each path. If these lengths can be easily computed then the number of calculations for an entire radiograph simulation can be reduced by a factor of several hundred.

The method used by the simulation package detailed in this report uses the latter approach. The package provides a number of base elements (spheres, rectangular solids, cylinders, et al.) which may be combined in arbitrary orientations to form more complex objects. We have derived analytic expressions for the lengths of intersection of arbitrary lines with these base elements, so the lengths required

in Eq. 2.4 can be quickly determined. This allows radiograph simulation in almost real time on a PC platform. The actual simulation time depends on object complexity and size. As an example, simulating a radiograph of the T-joint of Fig. 2.1 (see for example Fig. 4.5 or 4.6) on our experimental system (a floating point coprocessor equipped 33-MHz 386 PC-compatible microcomputer) requires 18 seconds. The simulated T-joint was built from 2 rectangular solids, 2 cut cylinders, 1 rectangular slot, and 1 cylindrical hole (the last two elements are simulated flaws).

2.2.1 2D Simulations

As an aid to understanding 3D simulations, let us first consider simulations in 2 dimensions. Consider a single ray incident through a disk of radius R centered at the origin, as illustrated in Fig. 2.2. The line $L = L_{\theta,w}$ can be specified by the angle θ it makes with the positive x-axis and by the distance w of L from the origin. In the figure, the line segments of length w and R form two sides of a right triangle (one leg and the hypotenuse). The length of the remaining side is $\sqrt{R^2 - w^2}$, which by symmetry is half the total length of the intersection of line L with the disk. Therefore,

$$\Phi(L_{\theta,w}) = 2\sqrt{R^2 - w^2}, \tag{2.5}$$

where $\Phi(L_{\theta,w})$ denotes the length of the intersection of the line $L_{\theta,w}$ with the disk.

This is a simple case due to the rotational symmetry of the disk, i.e., notice

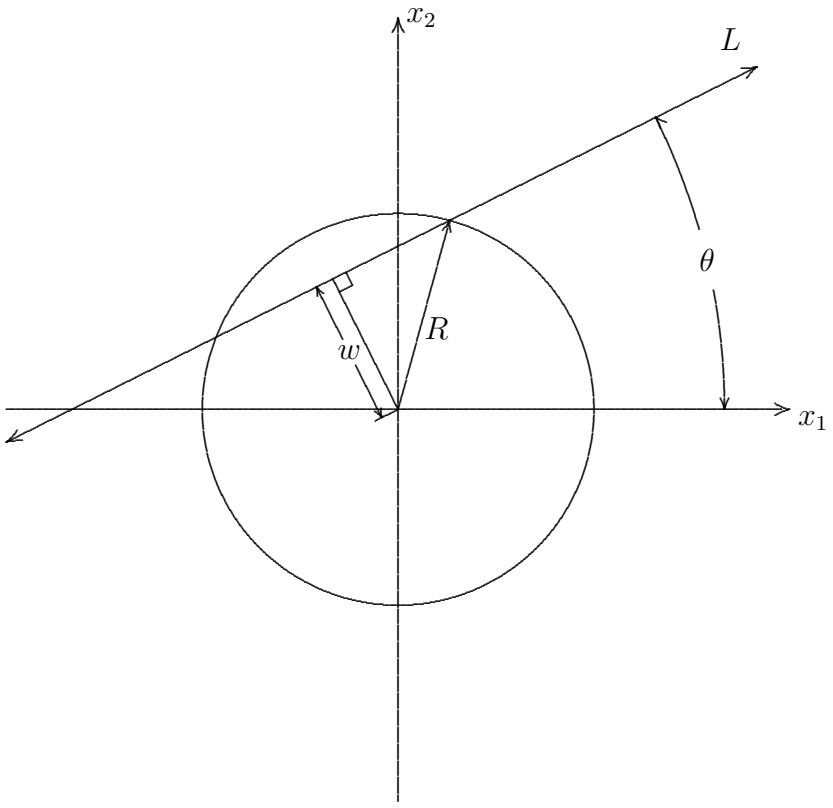


Figure 2.2: Illustration of intersection of line L with disk of radius R.

that Eq. 2.5 is independent of the angle θ. Let us next consider a more complicated situation, the intersection of a line with a rectangle centered at the origin, as in Fig. 2.3.

We could define the line L in terms of θ and w as in the previous example, but it will ease future considerations to define L using vector notation. Let $\vec{w}$ be the coordinates of the point on L that is nearest the origin (this makes $\vec{w}$ perpendicular to L). Next let $\vec{v}$ be a unit vector parallel to L. Then any point on L can be reached by starting at $\vec{w}$ and moving some distance (possibly a negative distance) in direction $\vec{v}$. Thus we can parameterize the line L by

$$(x_1(s), x_2(s)) = L_{\vec{v},\vec{w}}(s) = s\vec{v} + \vec{w}. \tag{2.6}$$

Formally, the conditions on $\vec{v}$, $\vec{w}$ are $\|\vec{v}\| = 1$ and $\langle \vec{v}, \vec{w} \rangle = 0$.

To calculate the length of the intersection of the line with the rectangle, it suffices to determine the parameters $s_{\min}$ and $s_{\max}$ at the points where the line intersects the sides of the rectangle. Since the parameterization is by arc length ($\|\vec{v}\| = 1$), the length of the intersection is just $s_{\max} - s_{\min}$.

The rectangle can be constructed as the intersection of 4 half spaces, i.e.,

$$\begin{aligned} R &= \{(x_1, x_2) : x_1 \le e_1\} \cap \{(x_1, x_2) : x_1 \ge -e_1\} \cap \\ &\quad \{(x_1, x_2) : x_2 \le e_2\} \cap \{(x_1, x_2) : x_2 \ge -e_2\}, \end{aligned} \tag{2.7}$$

where each half space is determined by one of the sides of the rectangle R. Since the points of L that lie inside the rectangle must also be in each of these half

spaces, we obtain the following 4 conditions on s:

$$-e_1 \leq x_1(s) = sv_1 + w_1 \leq e_1$$

$$-e_2 \leq x_2(s) = sv_2 + w_2 \leq e_2.$$

Solving yields

$$\begin{aligned} s_{\min} &= \max\left[\min\left(\frac{\pm e_1 - w_1}{v_1}\right), \min\left(\frac{\pm e_2 - w_2}{v_2}\right)\right] \\ s_{\max} &= \min\left[\max\left(\frac{\pm e_1 - w_1}{v_1}\right), \max\left(\frac{\pm e_2 - w_2}{v_2}\right)\right], \end{aligned}$$

which yields the length of the intersection of L with the rectangle as

$$\Phi(L_{\vec{v},\vec{w}}) = \max(s_{\max} - s_{\min}, 0). \tag{2.8}$$

While it is possible to derive this result by other means, this approach generalizes easily to more complicated geometries in 3 dimensions. One must only write down a system of equations defining the body geometry, and then apply those equations to the vector parameterization of the line (Eq. 2.6).

Of course, both of these examples assumed that the object (disk or rectangle) was centered about the origin. Let us now develop a method to handle the general situation. The technique may seem a bit excessive, but the generality is needed for extension to the 3 dimensional situation.

Refer to Fig. 2.4, which shows a line L intersecting a rectangle with an arbitrary orientation. This rectangle can be obtained from a rectangle in the standard

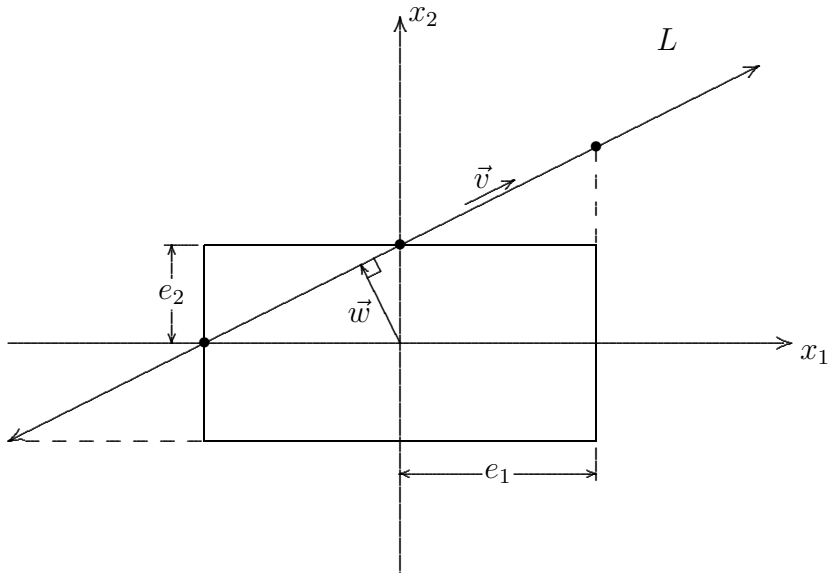


Figure 2.3: Illustration of intersection of line L with a rectangle of width $2e_1$ and height $2e_2$.

orientation of Fig. 2.3 by first rotating about the origin the amount θ in the counterclockwise direction, and then translating the rotated rectangle by $\vec{P}$.

Note that the equations defining Eq. 2.8 work for any line L. Thus, rather than attempting to generalize those equations for an arbitrary rectangle, we simply introduce a new coordinate system (the x_1', x_2' axis in Fig. 2.4) and transform the parametric equation for the line to the new coordinates. This requires operations on the line that are basically just the inverse operations of the rigid body motion applied to the rectangle.

Formally, let U be the orthogonal transformation given by

$$U = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}. \tag{2.9}$$

This provides a rotation in the clockwise direction by the amount θ. Then define T to be the rigid body motion

$$T\vec{x} = U(\vec{x} - \vec{P}). \tag{2.10}$$

Let $\vec{v}$ and $\vec{w}$ be the parameters defining the line in the original coordinate system (refer to Eq. 2.6), and let $\vec{v}'$ and $\vec{w}'$ be the parameters with respect to the new (rotated) coordinate system. Thus the line $L_{\vec{v},\vec{w}}$ in the original coordinate system becomes $L_{\vec{v}',\vec{w}'}$ in the new coordinate system. The new parameters are given by

$$\begin{aligned} \vec{v}' &= U\vec{v}, \\ \vec{w}' &= T\vec{w} - \langle T\vec{w}, \vec{v}' \rangle \vec{v}'. \end{aligned}$$

2.2.2 3D Base elements

We now turn to three dimensional simulations. As explained in the introduction, the simulation package works on objects built up from a handful of basic elements. The package currently supports 6 3-dimensional solid elements, listed in Fig. 2.5. Some of the base elements may be modified by the introduction of **cut-planes**. A cut-plane is a plane which intersects the base element, dividing it into two regions, only one of which is retained. Thus one can work not only with the specified base elements, but also with parts "cut-off" from those elements. Multiple cut-planes may be introduced as needed, allowing for very abstract shapes. In fact, one can

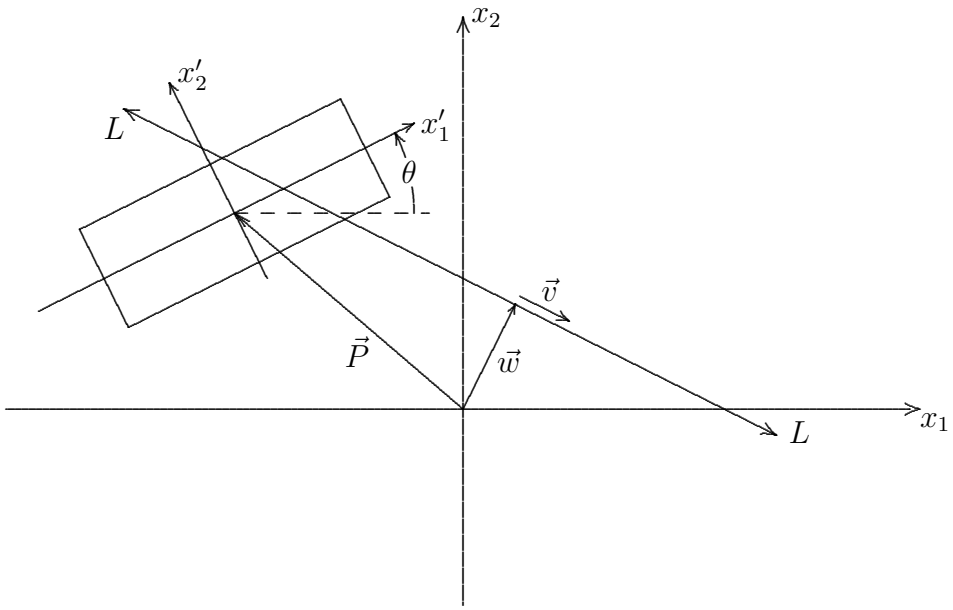


Figure 2.4: Illustration of intersection of line L with a rotated and translated rectangle.

BASE ELEMENT TYPES

Sphere
Ellipsoid*
Cylinder
Elliptical Cylinder*
Box*
Free Form*

Figure 2.5: The 6 building elements currently supported by the radiograph simulation package, where * indicates elements admitting cut-planes.

produce elements built from cut-planes alone. This is the "free form" base element. It starts out filling all of 3-space, with each cut-plane throwing out one half-space. For example, the "box" (parallelepiped) element type can be constructed as the intersection of 6 half-spaces.

Since the box element type can be constructed from the free form type with 6 cut-planes (one for each face), why is the box type included among the base element types? There are two reasons. One is convenience—it is easier to position a box as a single element rather than as 6 intersecting half spaces. But the other reason is computational speed. The box type is a specific alignment of half spaces which allows simplifications in the attenuation calculations. In fact, of the 6 base element types, only 3 are really necessary. The sphere is a special case of the ellipsoid type, the cylinder a special case of the elliptical cylinder, and the box is a special case of the free form type. But in the special cases, geometric considerations allow for decreased attenuation calculation time.

The interested reader is directed to Appendix A for program implementation

details.

2.2.3 Beam geometries

The simulation package currently provides parallel and (point) cone beam geometries. Each beam geometry has its own ray generation subroutine so that additional beam geometries (for example, a finite source beam) can be introduced as needed. Note, however, that complicated beam geometries can seriously increase program execution times.

To see the effects of the beam geometry on the radiograph simulation, refer to Fig. 2.6 through 2.8. Fig. 2.6 illustrates a simulated experimental setup, with rays from an X-ray source directed toward one edge of a triangular prism. Fig. 2.7 shows the simulation with a parallel beam geometry. This can be realized physically with a finite source located relatively far from the object. Notice that the radiograph simulation is darkest on the left side of the prism, where the X rays are attenuated along the entire length of the prism, and lightest on the right, where only a small amount of the prism intersects the X-ray beam.

Consider now Fig. 2.8, which is the same prism image via a simulated cone beam. This simulates the result from a micro-focus X ray source place near the prism. Notice that the shape of the image is no longer rectangular. This effect is caused by X rays that enter the front face of the prism but that, due to the angle to the source, exit through the top face of the prism before reaching the back plane.

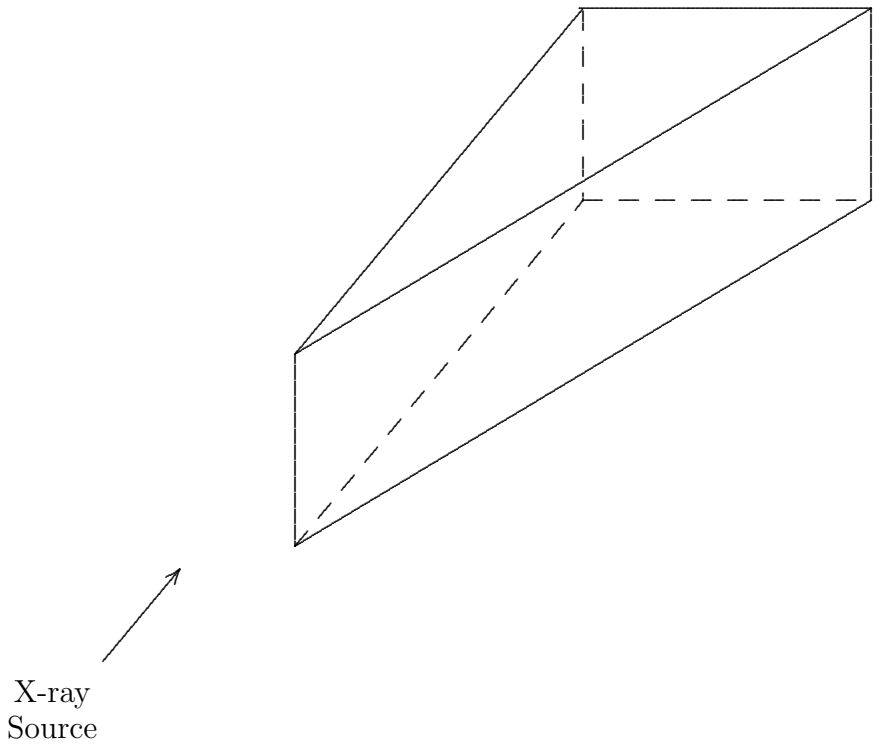


Figure 2.6: Triangular prism constructed as a "free form" element.

Also notice that the left side of the image is still darkest, but there is variation on this edge near the top and bottom (for the same reason).

2.3 Comparison of simulated and experimental radiographs

Let us consider the simulation of a sample we used in tomographic reconstruction experiments, a schematic of which is presented in Fig. 2.9. This sample is an aluminum cylinder (pipe) with a shaft of aluminum oxide. This shaft has two small cylindrical voids running the entire length of the shaft. The simulation data consists of 5 cylinder elements. The first two cylinders define the outer aluminum

Figure 2.7: Simulated radiograph of triangular prism with parallel beam geometry.

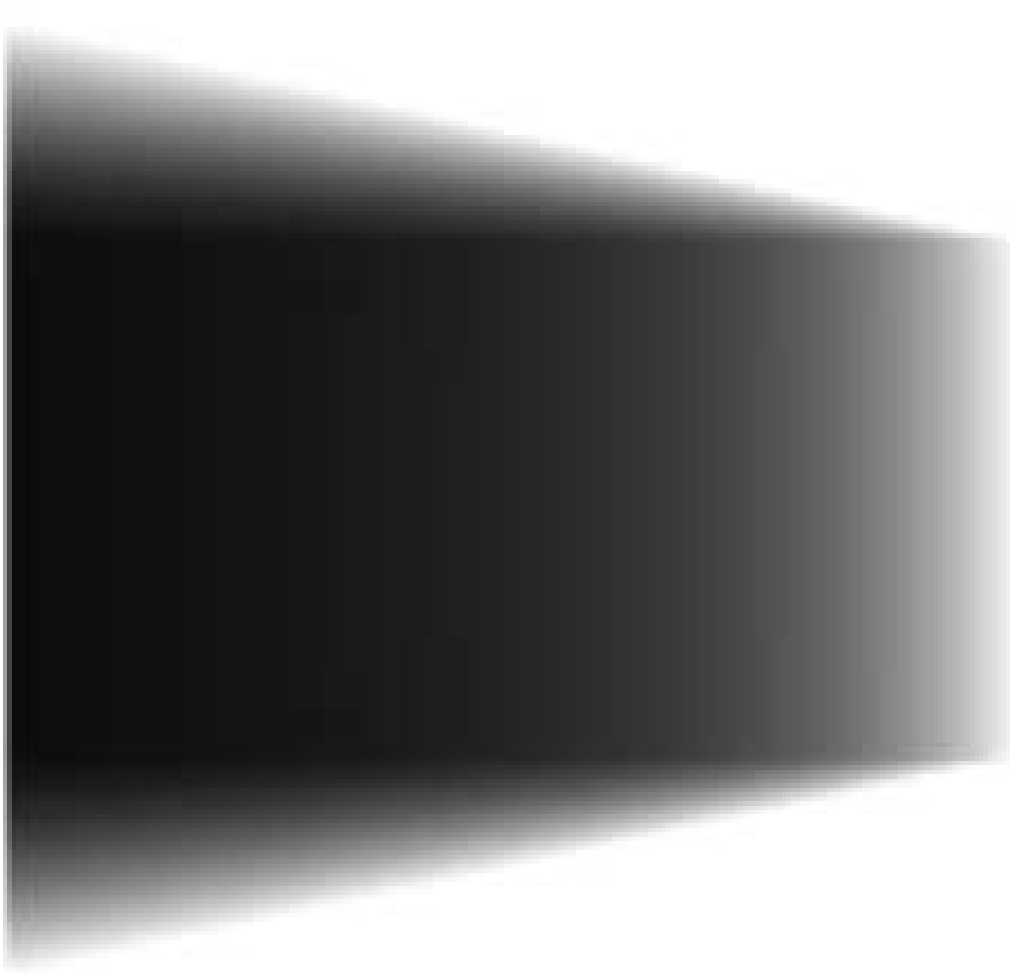


Figure 2.8: Simulated radiograph of triangular prism with cone beam geometry.

pipe, whereas the last three define the inner shaft with its two voids. The density of the aluminum oxide shaft (0.015) is slightly higher than the density of the outer aluminum pipe (0.01).

Fig. 2.10 shows three simulated radiographs taken from the side of the sample at three different angles (0°, 45°, and 90°). The axis of rotation is through the axis of the outer pipe, so the attenuation from this pipe is independent of the rotation angle. Notice how the inner shaft shifts to the left as the rotation angle increases. In the top image (0°), the voids in the inner shaft lie side by side relative to the display plane. The dark line through the center of the shaft shows the higher attenuation of X rays that pass between the two voids (as opposed to the X rays on either side which pass through the voids). In the bottom image (90°) the two voids are aligned with the X ray paths, resulting in a light strip through the middle of the shaft. These features are all visible in the actual real-time radiographic images of the experimental sample shown in Fig. 2.11. (The dark strips on the outside of the experimental radiographs is from lead shielding placed around the sample to reduce X-ray scattering.)

As mentioned, this sample was used in tomographic reconstruction experiments. In tomography, radiographs of a sample are obtained from many angles completely surrounding the sample. This data is then used to mathematically reconstruct the object. The number of required radiographs must be determined before data collection begins due to the impossibility of realigning the sample to its original

position with sufficient precision. If the number of directions is insufficient, then the reconstruction will suffer noticeable artifacts. On the other hand, one does not want to collect more data than is necessary. We determined the optimal number of directions with the aid of this radiograph simulation package. Fig. 2.12 shows a tomographic reconstruction (using the filtered backprojection algorithm) with simulated radiographs from 65 directions (spaced roughly 5.5° apart). Notice the banding inside the pipe and the serious reconstruction artifacts outside. Fig. 2.13 shows a nearly perfect reconstruction based on simulated radiographs from 315 directions (spaced slightly more than 1° apart). Fig. 2.14 shows the reconstruction from 315 actual real-time radiographic images.

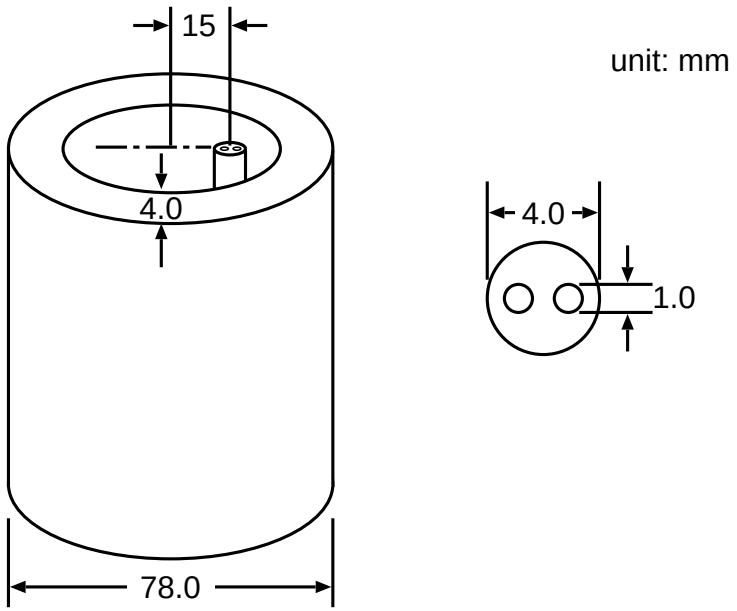


Figure 2.9: Schematic of experimental sample.

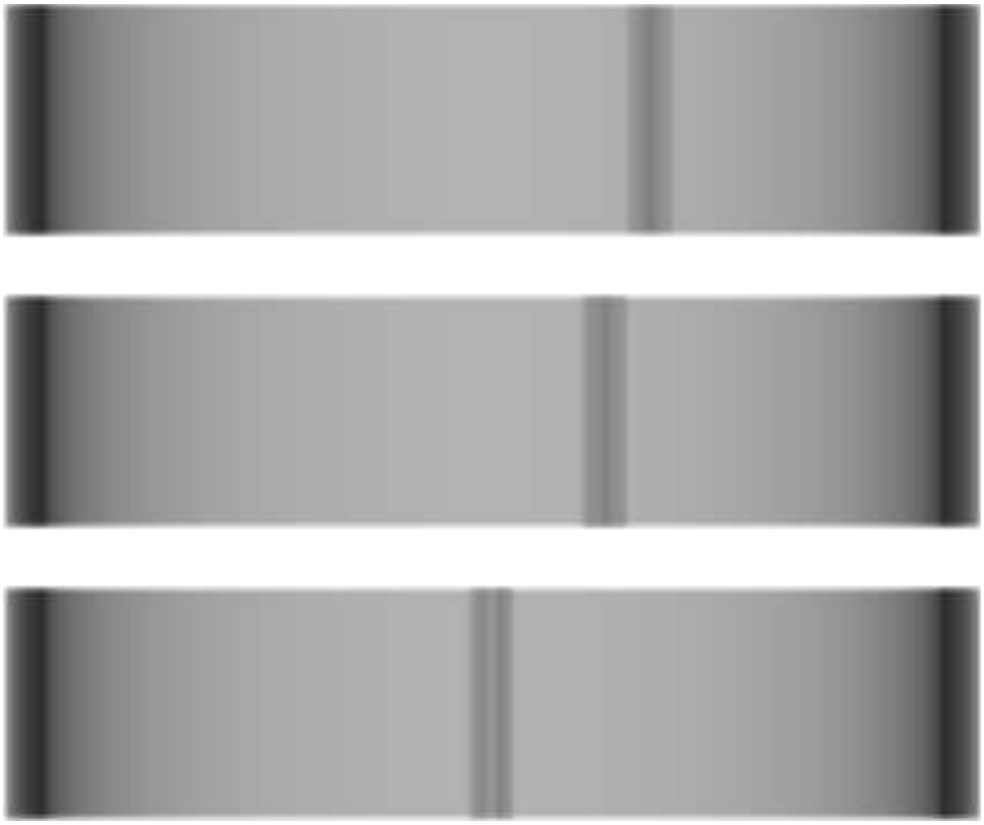


Figure 2.10: Simulated radiographs of the experimental sample: 0°, 45°, 90° (top to bottom).

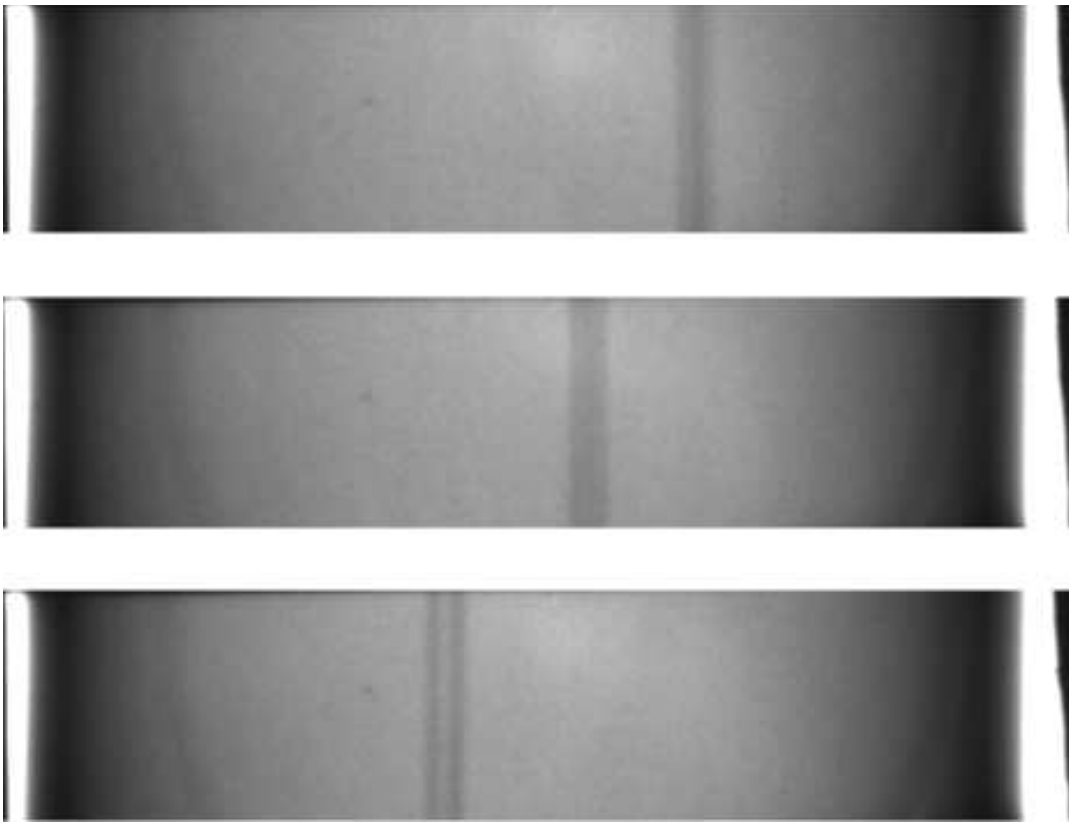


Figure 2.11: Actual real-time radiographs of the experimental sample: 0°, 45°, 90° (top to bottom).

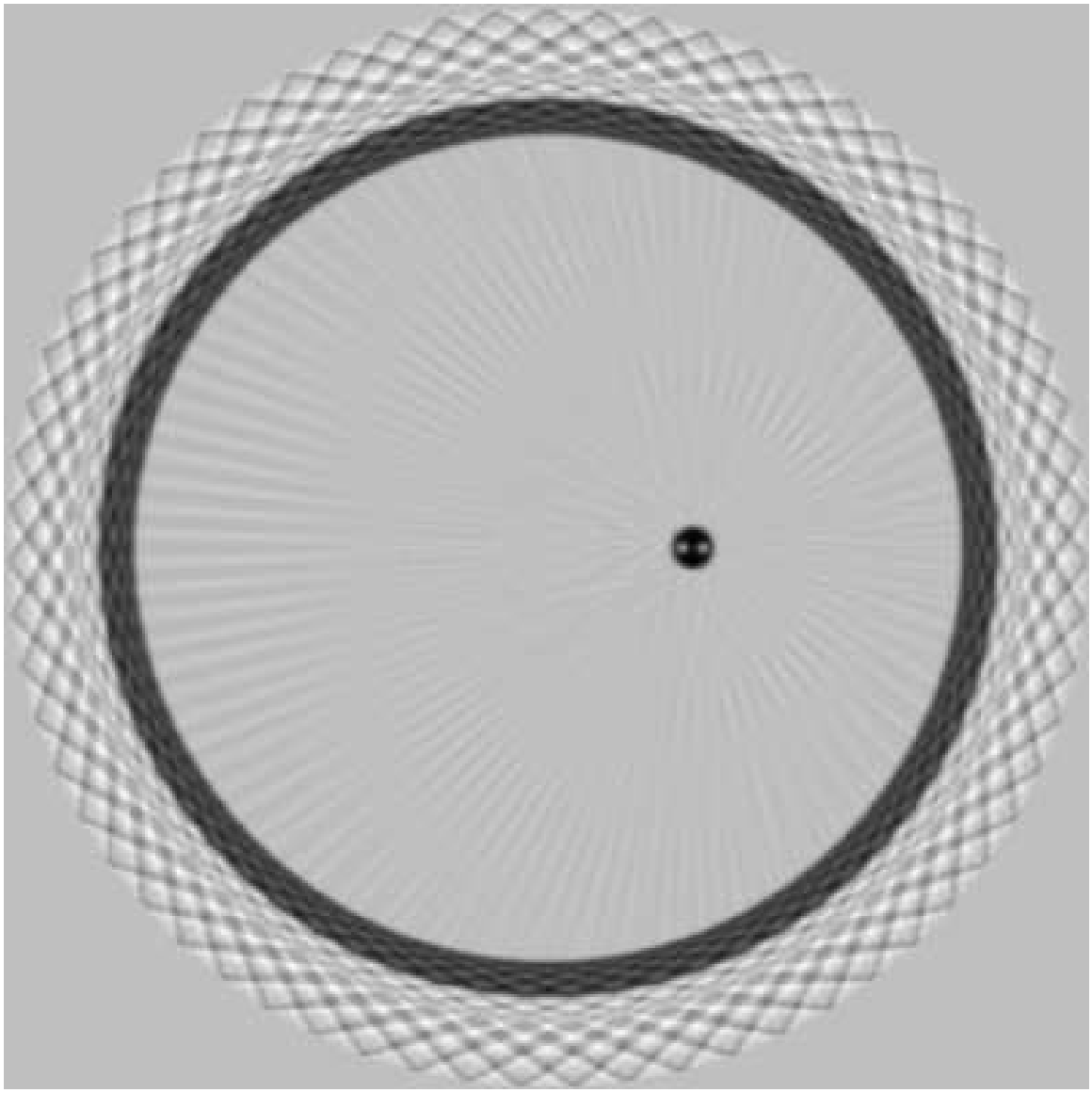


Figure 2.12: Tomographic reconstruction (FBP) from simulated data, 65 directions.

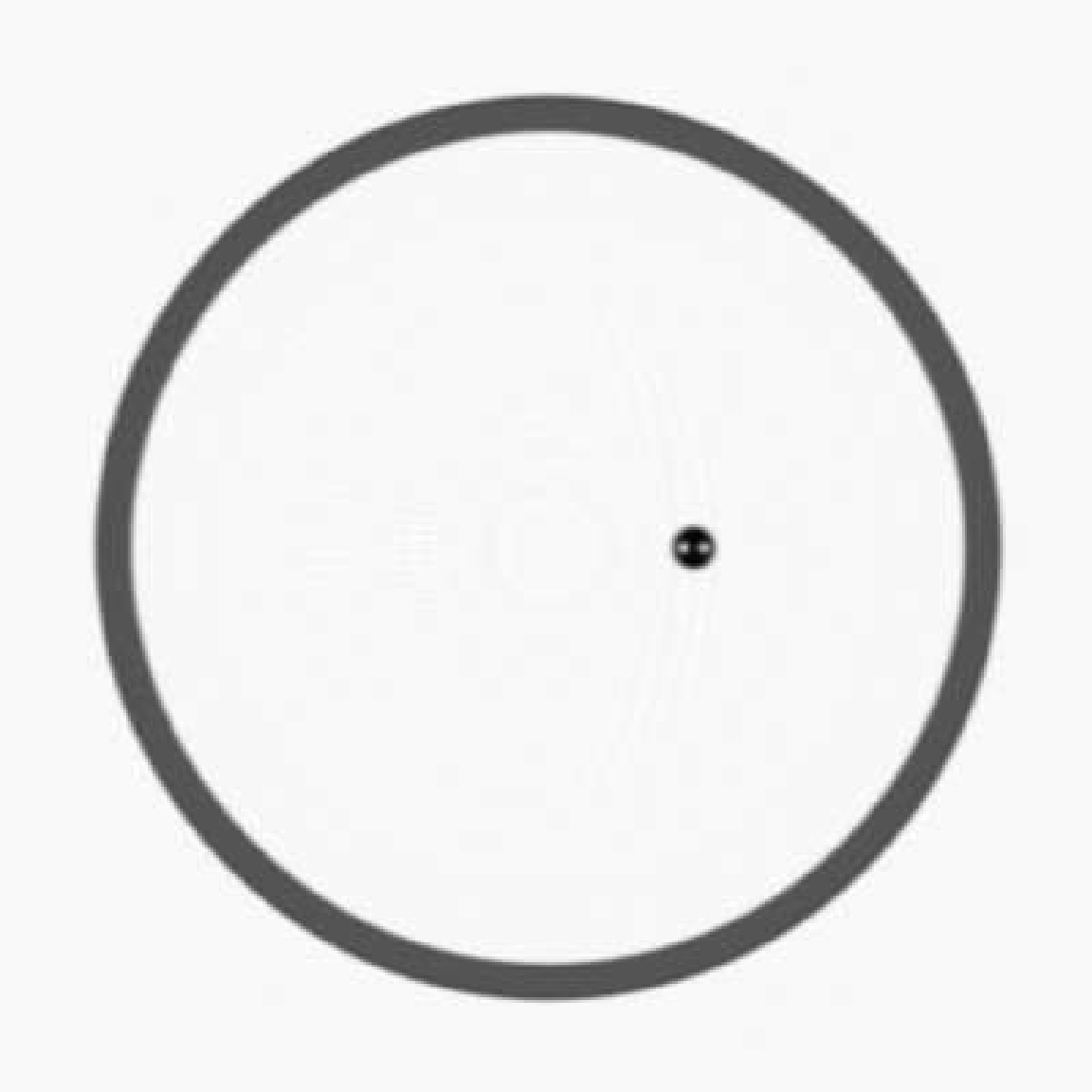


Figure 2.13: Tomographic reconstruction (FBP) from simulated data, 315 directions.

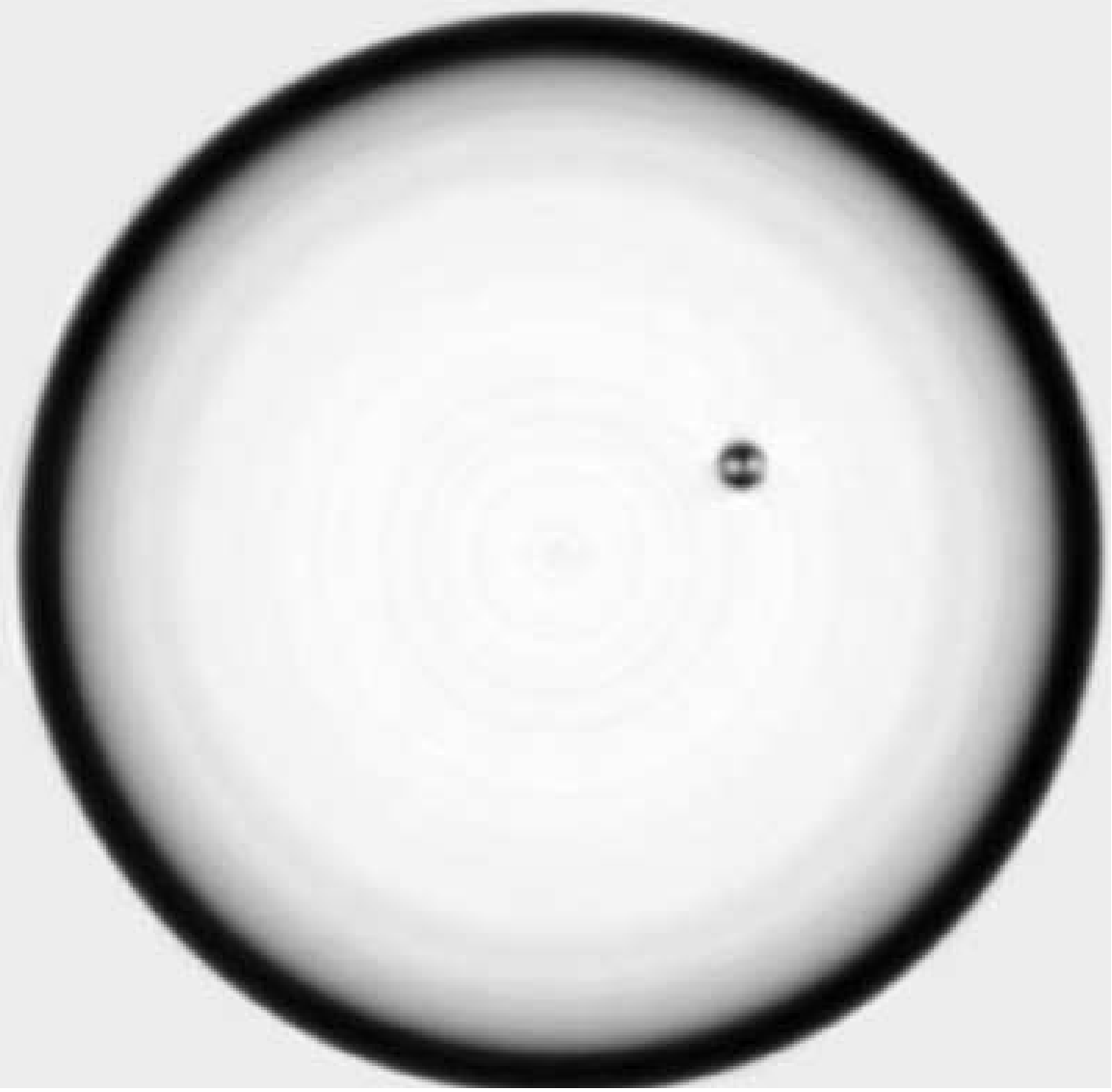


Figure 2.14: Tomographic reconstruction (FBP) from experimental data, 315 directions.

CHAPTER III

Cone beam tomographic reconstructions

This chapter describes an implementation of a 3D cone beam reconstruction algorithm based on some work by Feldkamp [3, 25]. The initial work on this topic in our research group was performed by Ligou Wang [26]. Presented also are results of applying this algorithm to both simulated and experimental data.

3.1 Theory

Consider the 3 dimensional analog to the tomographic reconstruction problem outlined in Chapter I. The body being examined is now 3 dimensional, the X-ray radiation fills a solid angle in 3 space, and the projection data is captured on a 2 dimensional detector array. The most common data collection geometry is to rotate the X-ray source around the body tracing out a planar circle. (Equivalently, the source may be held fixed and the body rotated about an axis perpendicular to the X-ray radiation.) If the X-rays are essentially parallel (for example, if the source is placed at a considerable distance from the object), then the reconstruction prob-

lem can be restricted to individual planes passing through the object, which are perpendicular to the axis of rotation. On each of these (stacked) planes one perform the tomographic reconstruction using the standard 2 dimensional techniques. Restacking these planar reconstructions results in a volume reconstruction.

There has been much work devoted to efficient implementation of such parallel beam 3 dimensional reconstructions. See for example [27, 28]. However, X-ray radiation from a physical source is never exactly parallel, and as the distance from the source to the object increases so must the X-ray energy. X-ray radiation scattering also becomes a larger problem. Moreover, with the advent of microfocus X-ray tubes the possibility of placing the source close to the object to take advantage of geometric magnification is enticing. In this situation the X-rays are no longer parallel, but rather spread out as a cone beam. Reconstructions with this type of scanning geometry is referred to as cone beam tomography.

There are theoretical difficulties with the cone beam scanning geometry. Let us call the midplane that plane through the object which contains all the source locations, i.e., the circle describing the motion of the X-ray source lies in the midplane. Then obviously object reconstruction in the midplane is equivalent to 2 dimensional fan beam reconstruction. Note, however, that there is no other plane which contains all the source locations, and therefore no 2 dimensional tomographic reconstruction algorithms can be applied outside the midplane. One straightforward possibility is to adapt an iterative technique (similar, for example, to the

one presented in Chapter IV) to this problem. However, with the amounts of data necessary to process for 3 dimensional reconstructions the time savings using a faster convolution technique could be considerable.

A convolution reconstruction method has been suggested by Feldkamp [3, 25]. (See also [2].) It is an extension of the 2D filtered backprojection method, which provides an approximate reconstruction outside the midplane. (On the midplane the reconstruction is identical to the filtered backprojection method.) Let $P_\theta(\rho, z)$ denote the (2D) projection of the object in the θ direction, and let D denote the distance from the source to the object. The first step is to adjust for distance variations in the projection data:

$$\tilde{P}_\theta(\rho, z) = \frac{D \cdot P_\theta(\rho, z)}{\sqrt{D^2 + \rho^2 + z^2}}. \tag{3.1}$$

Next convolve each level (fixed z) of the modified projection $\tilde{P}_\theta(\rho, z)$ by the usual FBP filter $h(\rho)$ in the ρ-variable. The filter $h(\rho)$ is the inverse Fourier transform of the truncated absolute value function, i.e.,

$$h(\rho) = \frac{1}{2} \int_{-W}^{W} |\omega| e^{2\pi i \omega \rho} \, d\omega. \tag{3.2}$$

The filtering is usually done in the (discrete) frequency domain, where convolution becomes multiplication, i.e., the discrete Fourier transform (in ρ) is taken of both $\tilde{P}$ and h and then the transformed functions are multiplied together pointwise. Applying the inverse Fourier transform to the product function yields the filtered, space domain projection $Q_\theta(\omega, z)$. This intermediate step is exactly the same as

for the standard 2 dimensional filtered backprojection algorithm, but it is done for **each** level z.

It remains only to do the backprojection. Here again some modifications are required to adjust for distance variations:

$$\alpha(s,t,z) = \int_0^{2\pi} \frac{D^2}{(D-t)^2}\, Q_\theta\Big(\frac{Ds}{D-t}, \frac{Dz}{D-t}\Big)\, d\theta, \tag{3.3}$$

where s and t are the rotated coordinates defined by

$$\begin{aligned} s &= x\cos\theta + y\sin\theta \\ t &= y\cos\theta - x\sin\theta. \end{aligned}$$

(This presentation is taken from [2].)

Generally the preceding calculations are done with z fixed to reconstruct one level of the object at a time. Then the individual level reconstructions can be stacked to form the desired 3D reconstruction. Note too that the formulae require that the point (x,y,z) be irradiated from **all** directions. Thus the volume that can be reconstructed depends upon the X-ray beamwidth. (To be precise, the volume that can be reconstructed is a sphere of radius $D\sin\Omega$, where Ω is half the radiation beamwidth.)

As mentioned previously, this reconstruction method is only approximate. It is exact in the midplane, but the accuracy drops as the angle from the midplane increases. (See [29].) The inaccuracy takes the form of blurring in the vertical (z)

direction. In particular, Feldkamp showed that the algorithm is exact if the object is homogeneous in the vertical direction [3].

Actually, any reconstruction method based on this scanning geometry will be imprecise outside the midplane. There is a reconstruction stability condition, known as Tuy's condition [30], which provides restrictions necessary for a stable reconstruction. (See also [31, 29].) Briefly stated, the requirement is that for a stable reconstruction **any** plane passing through the object must intersect at least one source location. Notice that in the scanning geometry described above that planes parallel to the midplane do not intersect any source locations. (The X-ray source locations all lie in the midplane.) To avoid this difficulty it is necessary to change the source scan location geometry from a simple circle, perhaps to a sinusoidal shape surrounding the object or alternately to two intersecting circles.

3.2 Results

To test the effectiveness of this algorithm and its implementation, we applied it to both simulated and experimental data. The experimental sample consisted of a hollow aluminum cylinder surrounding an aluminum oxide pipe running parallel to the sides of the cylinder. There were also two smaller holes inside the pipe running the length of the pipe. The diameter of the pipe is approximately 1/8th inch. X-ray projection data was collected from 315 equally spaced directions surrounding the object, with the midplane running perpendicular to the long axis of

the cylinder. The source to object spacing was 1.7 meters. Since the projection data was homogeneous in the vertical direction, the projection data was smoothed in the vertical direction to reduce noise and artifacts from the image intensifier.

One level of a reconstruction of the experimental sample is given in Fig. 3.1. (Due to the homogeneity of the experimental object in the z-direction, the level reconstructions are all identical.) The reconstruction array is 200x200 pixels. Note that the smaller holes inside the inner pipe are easily visible. Comparison to Fig. 3.2, which shows the analogous reconstruction from simulated data, shows that the reconstruction from experimental data has some blurring of the outer cylinder towards the inside of the object. Since this effect is not apparent on the reconstruction from the simulated data, it must be the case that this effect is not due to the reconstruction but rather due to the data.

Fig. 3.3 displays corresponding cross sections from the experimental and simulated data. Note that the experimental projection intensities do not grow as quickly inside the outer wall as do the intensities from the simulated data. Apparently this difference produces the wall blurring effect evident in the reconstruction from experimental data. It is not certain what the source of this difference is, but the prime candidate seems to be an effect known as beam hardening [32]. It is known that the attenuation of an X-ray beam as it passes through an object depends not only on the object but also on the energy of the beam. High energy beams are generally attenuated less than lower energy beams. Typical X-ray source

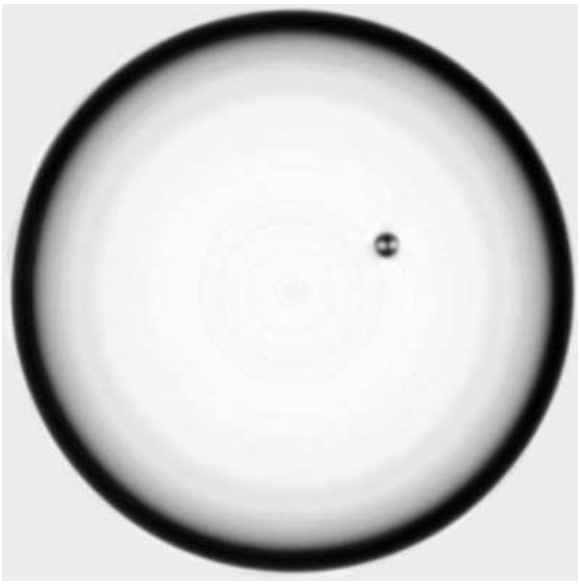


Figure 3.1: Tomographic reconstruction of experimental data using Feldkamp's cone beam algorithm.

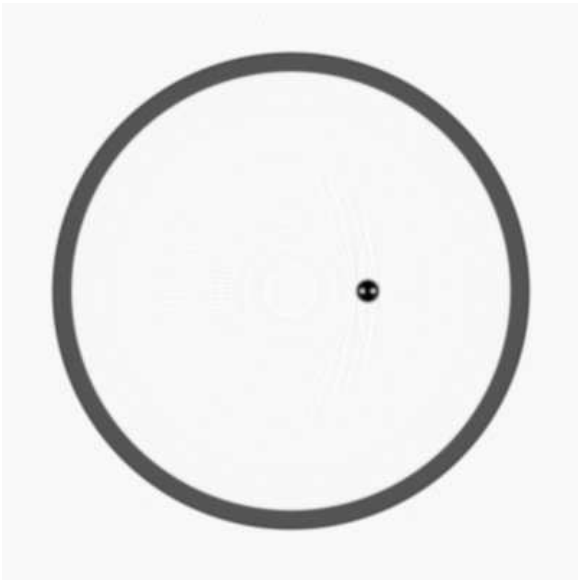
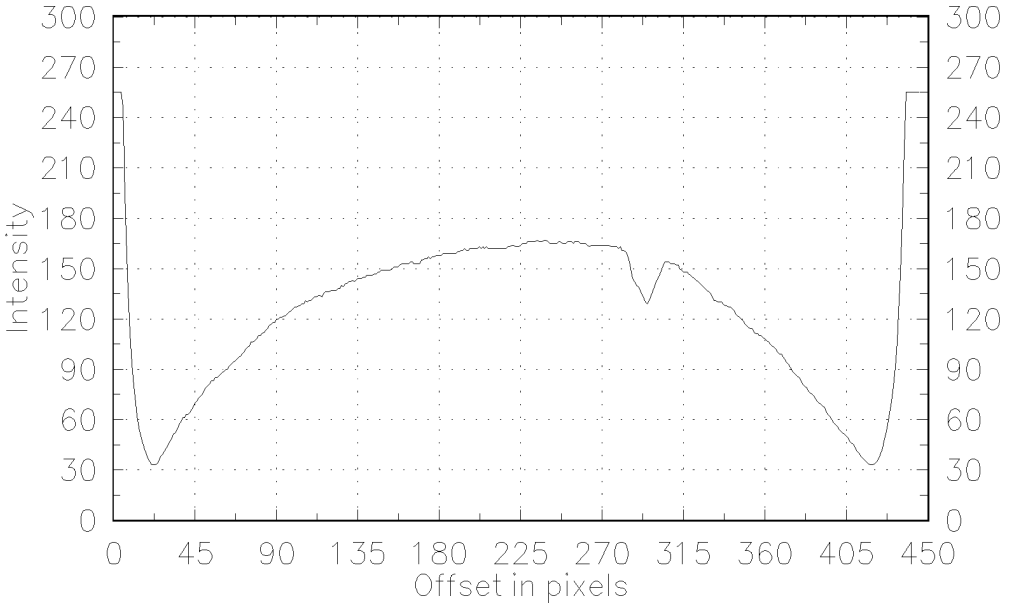
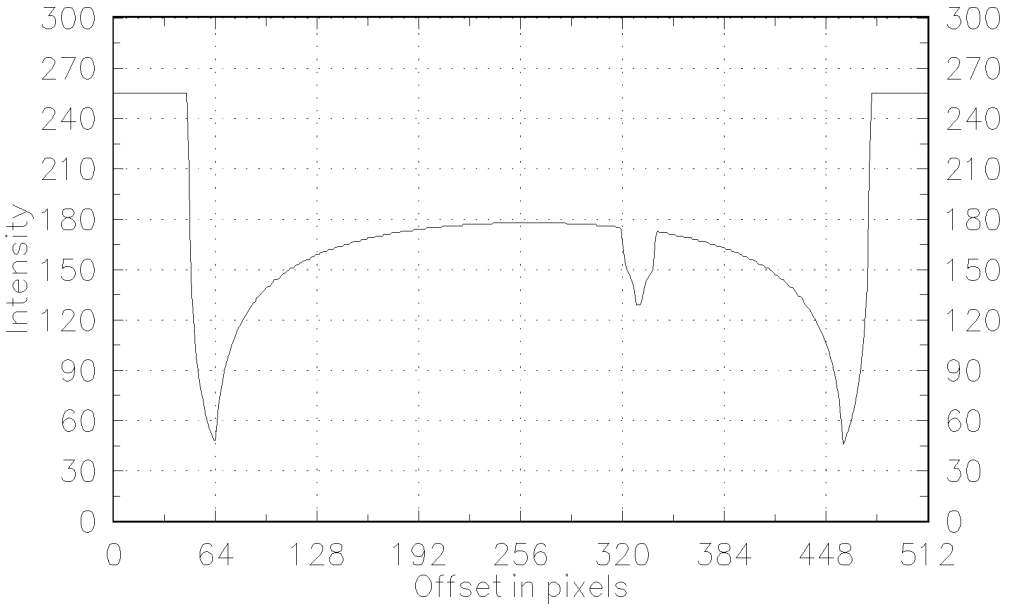


Figure 3.2: Tomographic reconstruction of simulated data using Feldkamp's cone beam algorithm.

are polychromatic in nature, that is to say the beam contains photons having a range of energies. The attenuation coefficient is no longer a function only of the position but also of the photon energy, so Eq. 1.2 no longer directly applies. It is possible to make partial corrections, see for example [33, 34], but a natural first step would be to try to introduce the beam hardening effect to the simulated data. This would produce a simulation more closely approximating actual experimental data and would provide understanding of the nature of the effect.



(a)



(B)

Figure 3.3: Graphs of the projection intensities for experimental data (a) and simulated data (b).

CHAPTER IV

Limited angle tomographic reconstructions

4.1 Data interpolation by analytic continuation

4.1.1 Theory

In practical tomography, the objects to be reconstructed are always of finite extent, i.e., there is some sphere of finite radius R such that the object in question can be completely contained inside it. From this condition it is easy to show that the Fourier transform of the attenuation function is analytic (holomorphic) in the corresponding frequency domain ($\mathbb{R}^2$ for planar objects, $\mathbb{R}^3$ for 3D objects). It is well known that analytic functions are uniquely determined by their values on any set containing a limit point. (See for example Theorem 10.18 of [35]). In the limited angle problem, each range of missing angles leads to a missing cone of data in the frequency domain. This is illustrated in Fig. 4.1. Therefore, it is theoretically possible to recover the missing data in the frequency domain by the method of analytic continuation. One such approach was suggested by Palamodov and Denisjuk in 1988 [4]. This was a short (3 page) mathematical paper without

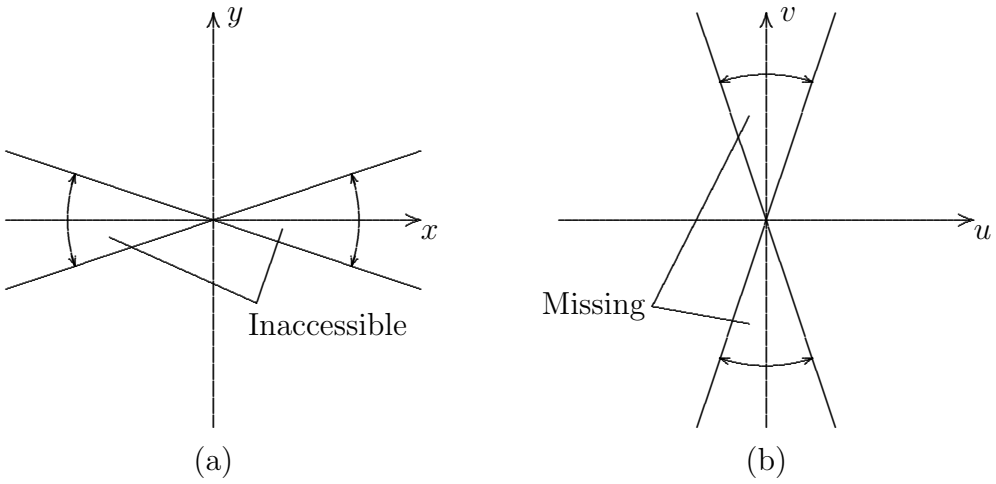


Figure 4.1: Illustration showing an inaccessible projection range (a) and the corresponding missing range in the frequency domain (b).

numerical implementation. The barriers to such implementation are nontrivial, and to the best of my knowledge our implementation is the first.

This approach uses a Cauchy integral along a line passing through the missing cone, as illustrated in Fig. 4.2. The missing region intersects the line as an bounded line segment. If we parameterize the line by arc length, and let the missing segment be denoted as $(-A, A)$, then the Cauchy integral can be written as

$$F(\omega_0) = \frac{e^{\sqrt{A^2-\omega_0^2}}}{\pi} \int_\Gamma \frac{\sin\left(\sqrt{x^2-A^2}\right)}{|x-\omega_0|} F(x)\, dx, \tag{4.1}$$

where F denotes the Fourier transform of the original attenuation function f, ω_0 represents a point on the integration line L in the missing range $(-A, A)$, and Γ represents the integration path consisting of the line L minus the interval $[-A, A]$.

In theory this technique allows the complete extrapolation of the missing data in the frequency domain. Once accomplished, the original attenuation function f can be recovered by simply taking the inverse Fourier transform of the recovered frequency function F. There are practical difficulties, however. First, the theory is for functions on the continuum, not for functions known only on a discrete grid that are the was actually available. (In fact, in theory the available finite sampling of the analytic function F does not determine F off the sample grid.) Also, the Discrete Fourier Transform (DFT) used to calculate the function F is only an approximation to the continuous space Fourier transform upon which the theory is built. The worst aspect of this approximation is that the DFT provides data on only a finite portion of the frequency domain. The approximation assumes that the function f is essentially bandlimited, i.e., that $F(\omega) \approx 0$ for $|\omega|$ large enough. But if the attenuation function f has sharp edges (common in industrial applications) then the magnitude of the corresponding frequency domain function F behaves asymptotically like $|\omega|^{-1}$, which is not insignificant in the context of Eq. 4.1. (Note that if one supposes that the attenuation function f **is** bandlimited, i.e., that $F(\omega) = 0$ for ω sufficiently large, then f no longer has finite extent and in fact F is no longer analytic!)

Another problem, possibly more serious, is the ill-posedness of the analytic continuation itself. As shown by Palamodov and Denisjuk [4], the integral of Eq. 4.1 is very sensitive to small changes in the function F along the integration

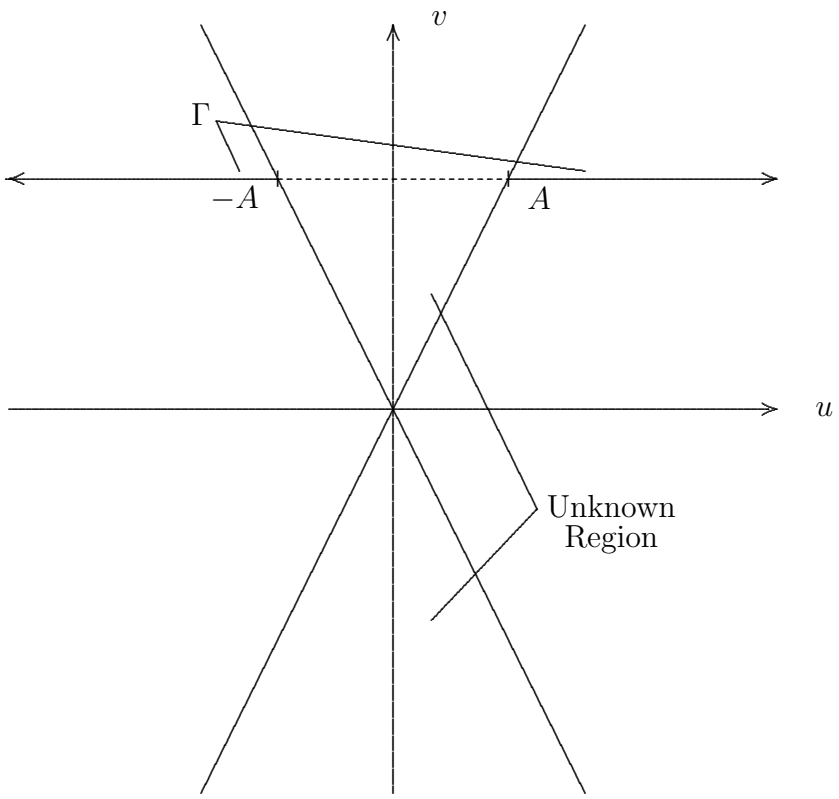


Figure 4.2: Illustration of the frequency domain showing the integration contour Γ for a Cauchy integral for data extrapolation in the missing region.

contour Γ. In particular, they show that the rate of growth of $F(\omega_0)$ inside the missing region can be exponential with respect to the size of the missing interval and data noise. This follows from the exponential term in front of the integral in Eq. 4.1.

The third difficulty in realizing Eq. 4.1 numerically is that the sampling grid for the frequency domain function F is polar, whereas the integral in Eq. 4.1 requires a rectangular grid. This difficulty occurs also in Direct Fourier Inversion methods for tomography. In fact, the success of the Filtered Backprojection method for tomography is attributable to its avoidance of the interpolation problem by performing the inverse Fourier transform using polar coordinates.

4.1.2 Implementation and results

The implementation of this algorithm was developed upon the filtered backprojection program available from the implementation of Feldkamp's cone beam algorithm. This is used to transform the projection data into the Fourier frequency domain via a 512 point FFT. The data in the frequency domain is in a radial format, whereas to evaluate Eq. 4.1 we need sample nodes along the integration contour. The more critical interpolation is in the radial direction. Zero padding the projection data causes the FFT to provide proper radial interpolation. For example, if the length of the projection data is double by the addition of zeros onto either end, then the FFT will provide a new frequency data sequence of double

the length of the unpadded sequence, with the new nodes interpolated between the previously existing nodes. If sufficient memory is available, the zero padding can be done to whatever length necessary to provide accurate interpolation. We chose an alternate approach which is slower but requires less memory. The FFT provides samples of the Fourier transform of the original data sequence considered as a summation of delta functions, i.e., suppose the space domain function f with samples $f(k) = f_k$ $(k = 0, 1, \ldots, N-1)$ is written as

$$f(x) = \sum_{k=0}^{N-1} \delta(x-k). \tag{4.2}$$

Here $\delta(x)$ represents the Dirac delta function at the origin. In this manner the values F_j of the sequence obtained from the FFT can be considered samples of the Fourier transform of f, namely

$$F(j/N) = \sum_{k=0}^{N-1} e^{-2\pi ikj/N}, \tag{4.3}$$

and in general

$$F(\omega) = \sum_{k=0}^{N-1} e^{-2\pi ikx/N}. \tag{4.4}$$

(Scaling factors have been ignored here as they differ depending on the form of the FFT being used.)

This method allows the placement of interpolated nodes at any place along any of the radial lines along a measured projection direction. The integration path is perpendicular to the bisector of the missing angular range, so the interpolated

points can be made to lie on this path with increasing frequency as the path approaches the missing range. The integral was numerically evaluated using the trapezoidal method. This method proved adequate with simulated data, and it was felt that higher order methods would be unstable with experimental data. Moreover, it is known that in tomographic applications Fourier frequency domain interpolations are generally more sensitive to radial interpolation than to angular interpolation [14].

Careful attention must also be paid to ensure the function f meets the requirement necessary for Eq. 4.1 to be valid. First, it is necessary that f be zero outside the interval $[-1, 1]$. This is a simple matter of rescaling, but it is critical that it be done. If this condition is violated, then Eq. 4.1 fails miserably.

Another problem, one that is not directly addressed by Palamodov and Denisjuk, is the fact that the FFT is equivalent to Fourier transform applied to delta functions. However, the Fourier transform of a delta function is a non-decaying sinusoid, and so the integral in Eq. 4.1 does not converge. To solve this problem I introduced the novel idea of convolving the delta functions in the space domain with the characteristic function of a small interval. This is equivalent to multiplying the Fourier transform of the delta functions by a "sinc" function, i.e., $\sin(ax)/ax$ (the parameter a depends on the size of the support for the characteristic function). When combined with the integral kernel of Eq. 4.1 the rate of decay of the integrand is of order x^{-2}, and so convergence is achieved. This also has the

effect of reducing the weight of the high frequency components in the integrand (x here corresponds to frequency, and large x corresponds to high frequencies). This is a desirable attribute since experimental noise tends to be disproportionately large in the high frequencies. This effect can be increased by multiplying in the frequency domain a second time by $\sin(ax)/ax$, forcing the tail of the integrand to decay like x^{-3}. However, recall that such multiplications are equivalent to convolving the original delta functions in the space domain by characteristic functions of intervals. The original delta functions are a pulse train of length N, which have been scaled to fit inside the interval $[-1, 1]$. Each convolution blurs the pulses and extends the nonzero function support by the size of the support interval for the characteristic function. For example, if we scale the pulse train to fit inside the interval $[-0.5, 0.5]$, then we can convolve with the characteristic function of the interval $[-0.125, 0.125]$ no more than twice. These are in fact the values I settled upon by experimental verification.

The exponential term in front of the integral prevents the use of this method if the missing region is too large. Note, however, that the exponent depends upon $A^2 - \omega_0^2$, where ω_0 is the point where the interpolation is desired. So even if the missing range $[-A, A]$ is large, the exponential term is manageable if the point ω_0 is close to the boundary. In this manner values for F can be computed near the boundaries of the missing interval $[-A, A]$, and then these new values can be used to decrease the size of the unknown region. This type of iterative method can be

used to extend the size of missing region that can be handled. However, any errors in the interpolation get magnified each successive step, so there is still a bound on the size of missing region that can handled. Using simulated data we found our implementation could manage intervals $[-A, A]$ for A not larger than 10. (Note the scaling units here are fixed by the fact that the space domain function needs to be scaled to fit inside the interval $[-1, 1]$.)

Obviously this puts a bound on the size of the missing angle that can be handled by this method. The bound depends not only on the size of missing interval that can be managed, but also on the range of frequency components that need to be recovered. Our experiments have shown that recovering the lower 20% of the frequency response in the Fourier frequency domain will provide a satisfactory tomographic reconstruction. With this criterion the largest missing angle that we can handle is 20°.

We tested our implementation on our experimental data by removing the projection data from the 20° range centered about the vertical axis. Note that projections from this direction are required to distinguish between the two smaller holes inside the post. Fig. 4.3 shows the reconstruction (using the filtered backprojection method) without any interpolation of the missing data, while Fig. 4.4 shows the reconstruction using the analytic continuation method. Note that although the second image has a horizontal sinusoidal artifact, it also has better shape definition around the borders and slightly better definition of the small holes inside the

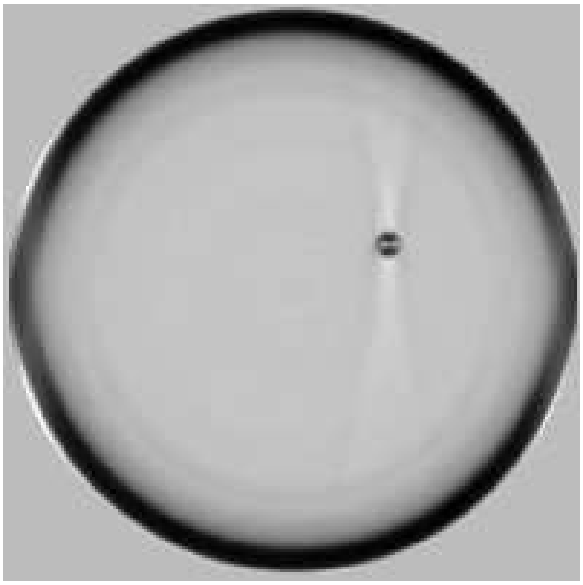


Figure 4.3: FBP reconstruction of experimental data missing projection data from 20° about the vertical axis.

post. I believe that more work on the algorithm will allow for the removal of the artifact and general resolution improvement.

4.2 Filtered backprojection reconstruction using a priori information

Instead of trying to use the existing data to interpolate data in a missing range, we can replace the missing data with a priori information. For example, let us consider the T-joint presented in the radiograph simulation section of this dissertation, Fig. 2.1. A top view of this T-joint is shown in Fig. 4.5 (actually a simulated radiograph from the top). For this simulation two flaws have been introduced: a rectangular slot between the beams and a cylindrical flaw in the flange material on one side. The flange material has been modeled as quarter cylinders introduced as

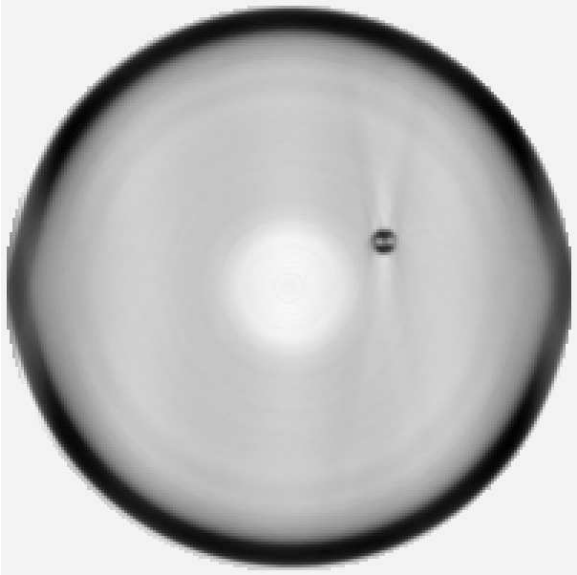


Figure 4.4: Reconstruction of experimental data missing projection data from 20° about the vertical axis. This reconstruction used the method of analytic continuation to interpolate the missing data.

elliptical cylinders with two cut-planes.

Fig. 4.6 shows simulated radiographs at 45° increments. You should be able to imagine the rotated T-joint and verify these simulations. We produced simulated radiographs of this T-joint from 315 equally spaced directions and performed tomographic reconstruction. The result is shown in Fig. 4.7. Note the reconstruction artifacts at the corners of the beams. This is related to the high frequency content of the image at the corners. Notice, however, that the introduced flaws are clearly visible, even though they cannot be detected in the individual radiographs. (This is one of the advantages of tomographic reconstruction.)

Next suppose that we cannot access the T-joint along the long end. Fig. 4.8

shows the tomographic reconstruction resulting from the omission of the radiographs in the ranges 80°–100° and 260°–280°. Notice that the introduced flaws are visible, but serious artifacts have been introduced. Next we replaced the missing data with data from an "ideal" T-joint. The ideal joint had neither flaws nor flanges. Here only the lower 20% of the frequency components from the ideal data were used. Fig. 4.9 shows the resulting reconstruction. The checkerboard banding is a result of the frequency limitation on the introduced ideal data.

4.3 Iterative reconstruction using a priori information

4.3.1 Theory

The iterative reconstruction method is also known as the algebraic reconstruction technique (ART). Specialized variants such as SIRT and SART differ on the handling of discretization details [2]. (The method used in the work presented below is essentially SART. Refer to either [14] or [2] for details.) The approach is based on a scheme of Kaczmarz [23] for the solution of a system of linear equations. The solution set to each equation is an affine hyperplane in some Hilbert space. If there is a solution to the system of equations, then the intersection of the solution sets for each equation is nonempty. Fig. 4.10 illustrates a simple example of this procedure. Each line represents a solution set for one of four linear equations. An initial point, f_0, is chosen, perhaps arbitrarily. The first iterate, f_1, is given by the orthogonal projection of f_0 onto the solution set S_1. The second iterate, f_2, is given

Figure 4.5: Simulated T-joint, top view.

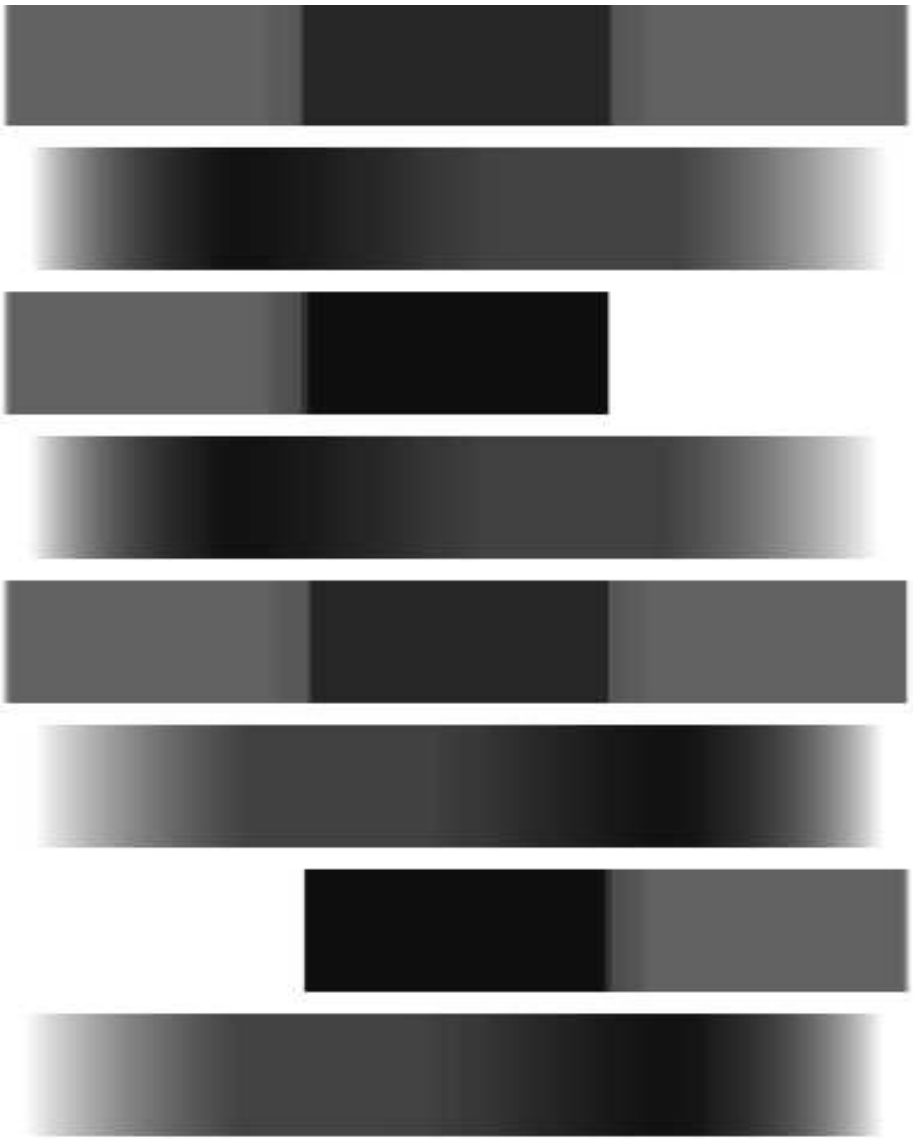


Figure 4.6: T-joint simulated radiographs at 45° increments.

Figure 4.7: T-joint reconstruction from simulated data, 315 directions.

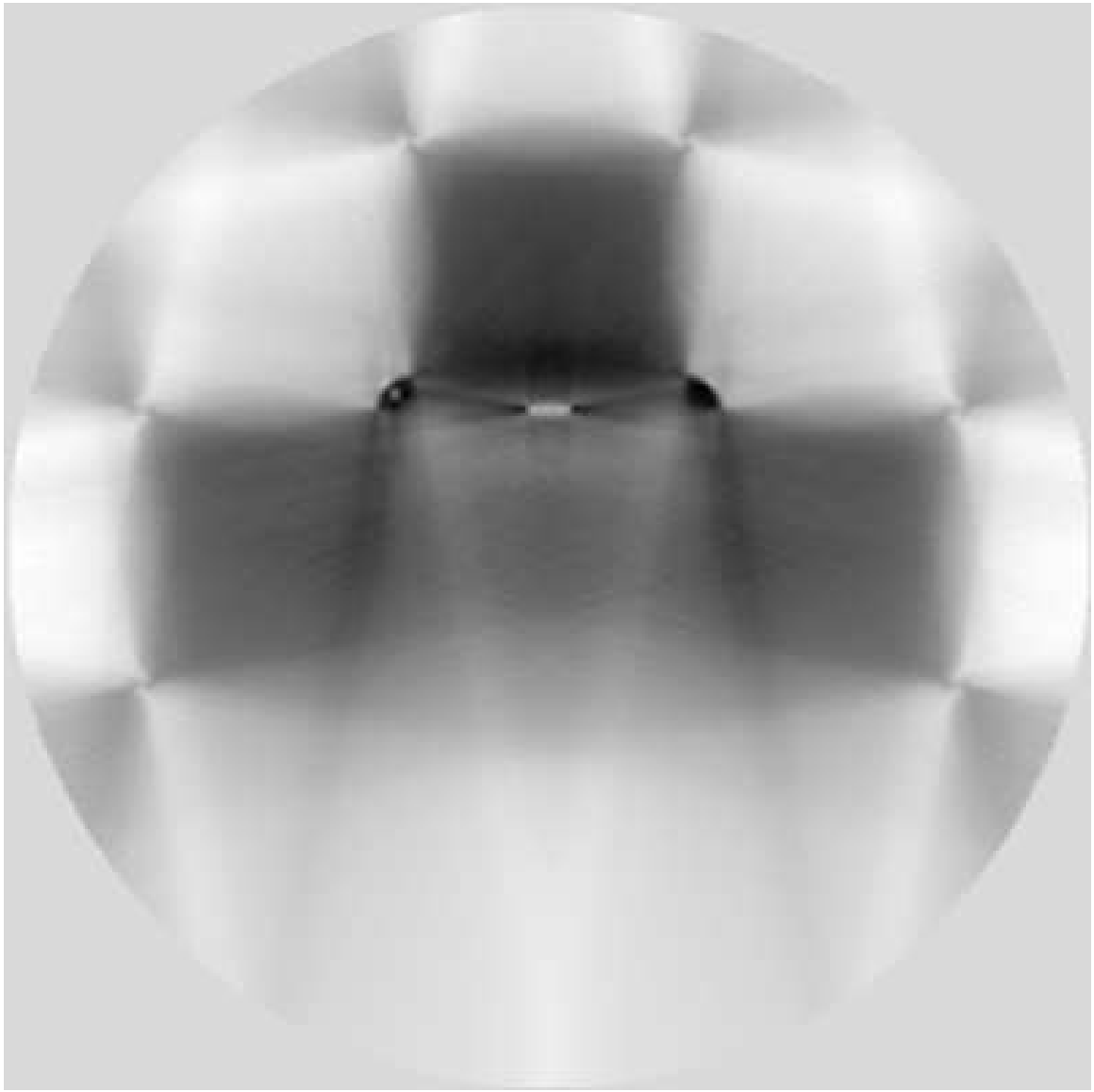


Figure 4.8: Reconstruction from simulated data, 20° missing angle.

Figure 4.9: Reconstruction from simulated data, 20° missing angle, lower 20% of frequency range filled with ideal data.

by the orthogonal projection of f_1 onto the second solution set, S_2. In general, the iterate f_k is given by the orthogonal projection of f_{k-1} onto the solution set $S_{k \bmod N}$, where N is the total number of equations (and therefore solution sets).

In this simple example, the solution sets are lines which lie in the plane $\mathbb{R}^2$, and the equations are linear equations in two variables. For tomographic reconstructions the setting is more complicated. The solutions are functions on $\mathbb{R}^2$ (or $\mathbb{R}^3$) corresponding to possible reconstructions, and the equations are determined by the projection information, so the solution space is infinite dimensional. Suppose the projection data P_k from the direction θ_k is given. Then the solution set S_k is the set of all functions h satisfying

$$\mathrm{Proj}_{\theta_k}(h) = P_k.$$

To be precise, let us restrict our set of allowable solutions to the square integrable functions h satisfying $h(x) = 0$ for all $|x| > 1$. (This is written $h \in L^2(\Omega)$ where Ω denotes the unit disk in $\mathbb{R}^2$.) The operation

$$\langle g, h \rangle := \int_\Omega g \cdot h$$

provides an inner product on this space of functions. This inner product is used to define the orthogonal projection onto the solution space S_k, namely

$$f_{k+1} = f_k - \mathrm{Smear}_{\theta_{k+1}} \left(\mathrm{Proj}_{\theta_{k+1}}(f_k) - P_{k+1} \right). \tag{4.5}$$

The operator $\mathrm{Smear}_{\theta_{k+1}}$ "smears" a projected function of one variable back across

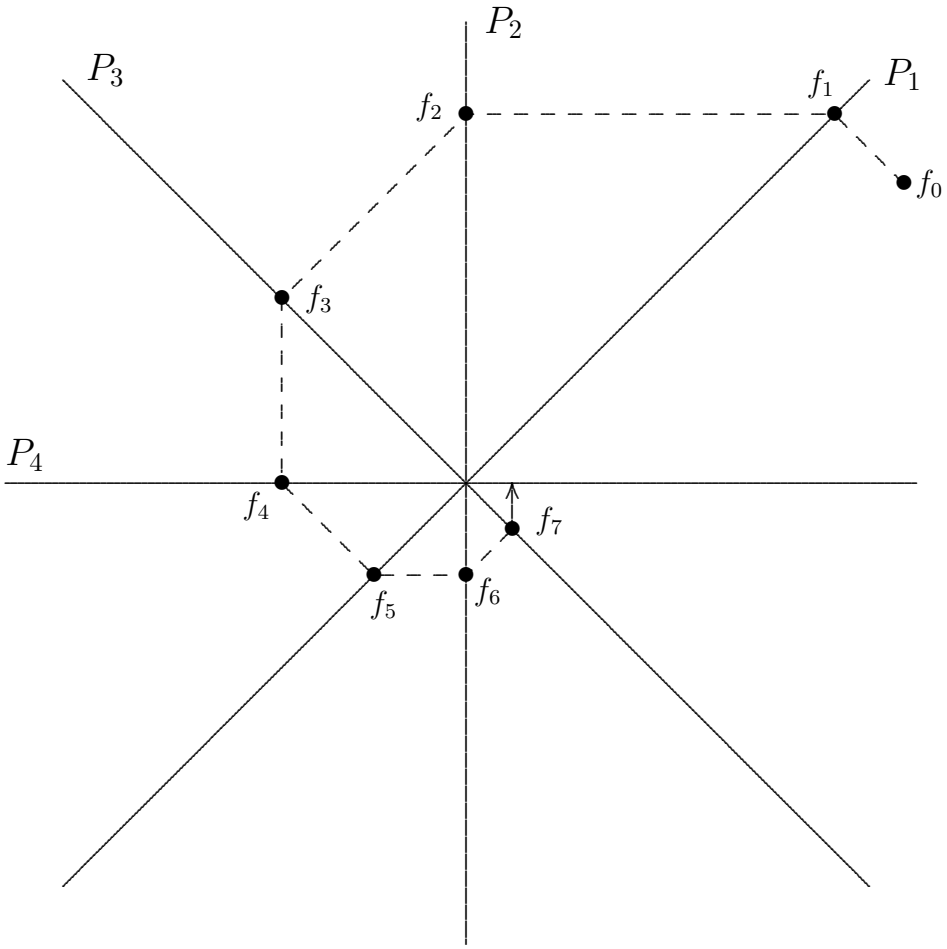


Figure 4.10: Graphic illustrating the convergence technique upon which the iterative reconstruction method is based.

Ω to produce a function that is constant along lines parallel to θ_{k+1} and satisfying

$$\text{Proj}_{\theta_{k+1}}(\text{Smear}_{\theta_{k+1}}(Q)) = Q.$$

It follows that $\text{Proj}_{\theta_{k+1}}(f_{k+1}) = P_{k+1}$, i.e., $f_{k+1} \in S_{k+1}$.

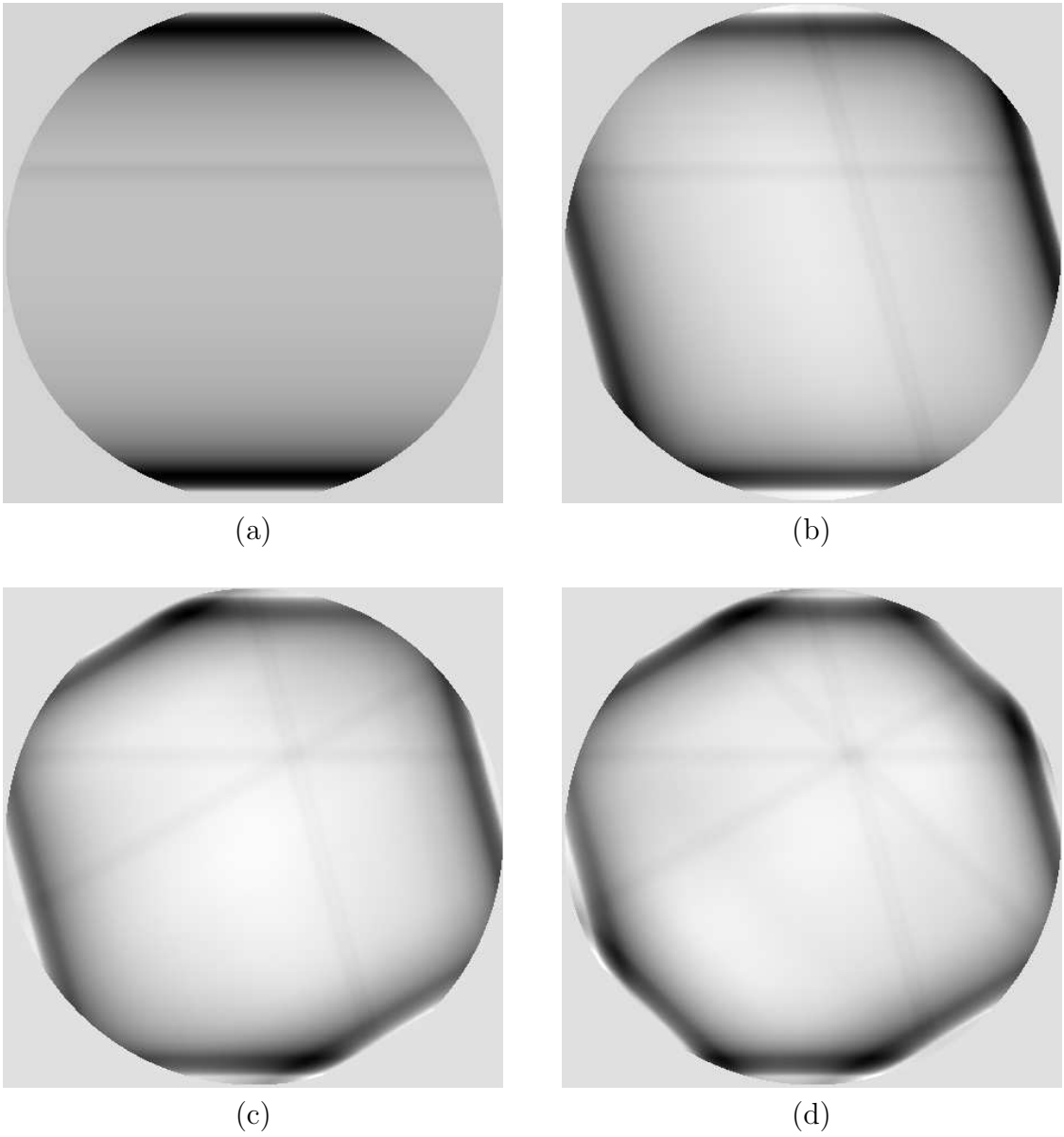
This forms the theoretical basis for the iterative reconstructive method. Fig. 4.11 illustrates the results obtained for the first few iterates for the simulated sample.

4.3.2 Inconsistent data

If each projection P_k is the precise projection from an actual function f, then the intersection of the solution sets S_k will contain at least one point, and the above iterative scheme will converge to the point in the intersection $\cap_k S_k$ that is closest to the initial iterate f_0 [14]. In practice, however, noise and experimental error cause the projections to be slightly incorrect, and in this situation it is likely that the intersection $\cap_k S_k$ will be empty. Fig. 4.12 illustrates this situation in 2 dimensions. Note that the iterative scheme no longer converges, but rather circles around a range of values. This problem is generally handled by introducing a relaxation parameter to Eq. 4.5, like so:

$$f_{k+1} = f_k - \lambda \cdot \text{Smear}_{\theta_{k+1}} \left(\text{Proj}_{\theta_{k+1}}(f_k) - P_{k+1} \right), \tag{4.6}$$

where the relaxation parameter λ is chosen between 0 and 2. It can be shown that if the projections are consistent, then the relaxed iteration converges to the same point as before (i.e., $\lambda = 1$). However, the smaller the value assigned to λ, the



(a) (b)

(c) (d)

Figure 4.11: Illustration of the first four iterates of the iterative reconstruction using the simulated projection data. The 0 function was selected as the initial iterate f_0. (a) f_1, $\theta_1 = 0°$. (b) f_2, $\theta_2 = 105°$. (c) f_3, $\theta_3 = 210°$. (d) f_4, $\theta_4 = 315°$.

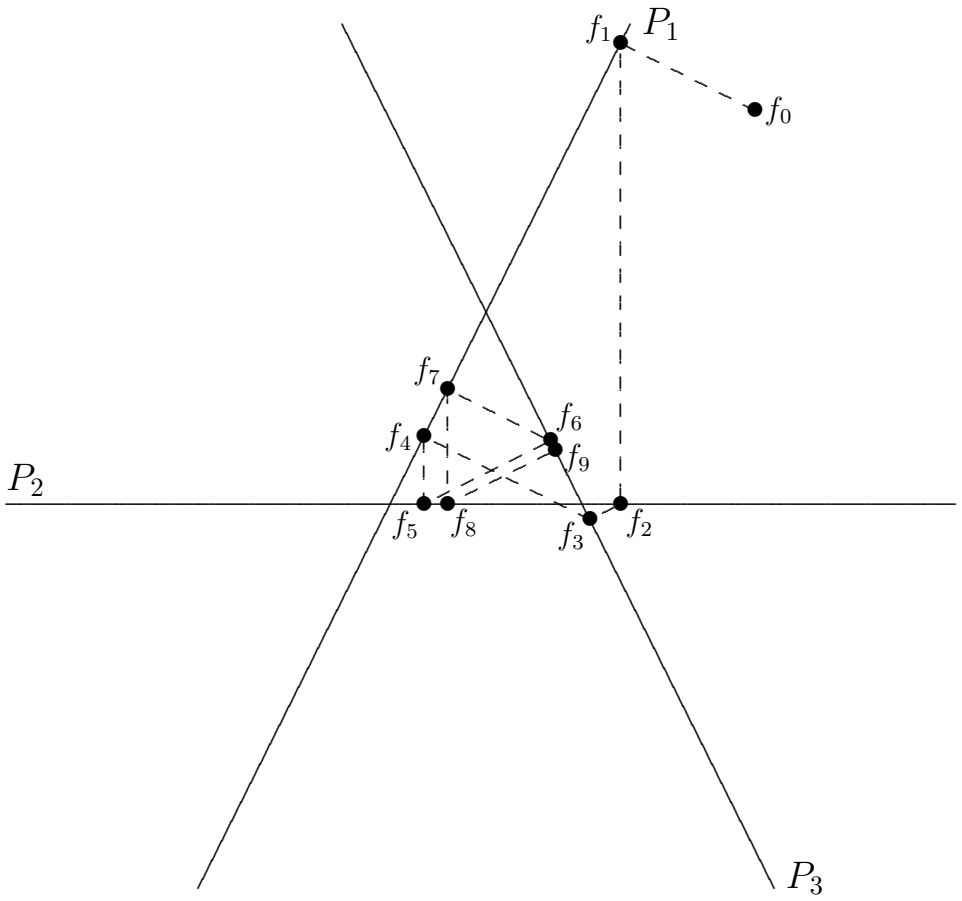


Figure 4.12: Graphic illustrating the convergence technique upon which the iterative reconstruction method is based.

slower the convergence. If the data is inconsistent, then for fixed λ the iterations still do not converge, but the region around which the iterates circle is decreased.

Practical reconstructions typically assign a very small value to λ, usually on the order of 0.05. Although the iterations do not converge completely, this value of λ is small enough to force the size of the oscillations down to the point where practical convergence is achieved. However, a value of λ this small also dramatically slows the convergence. So in the present work a different approach was taken. For the

first iterate λ is set to 1, and for each successive iterate the value of λ is decreased towards 0. One simple way to achieve this is by multiplying λ by a constant slightly smaller than 1 after each iteration:

$$f_{k+1} = f_k - r^k \cdot \text{Smear}_{\theta_{k+1}} \left(\text{Proj}_{\theta_{k+1}}(f_k) - P_{k+1} \right). \tag{4.7}$$

For the work presented in this paper r was set to 0.998. The data sets worked with contain 315 projections, so the relaxation value after one complete cycle through the projections is $r^{315} = 0.532$. This method allows for quick convergence and still guarantees convergence for inconsistent data.

4.3.3 Projection ordering

Another matter to be considered is the ordering of the angles θ_k. For example, the work presented in this paper used 315 equally spaced angles, at 0°, 1.14°, 2.29°, etc.. The natural ordering is $\theta_1 = 0°$, $\theta_2 = 1.14°$, and so on. However, if this ordering is used then the projections P_k and P_{k+1} are nearly identical, and so there is little movement between the iterates f_{k+1} and f_{k+2}.

To be rigorous, the similarity of two projections P_k and P_{k+1} corresponds to the angle between the solution spaces S_k and S_{k+1}. The angle between these subspaces (as affine subspaces of the underlying Hilbert space $L^2(\mathbb{R}^2)$) is *not* the same as the spacing angle between the projection directions θ_k and θ_{k+1}. The actual relation between the angular spacing Δ, taken in the range $[0, 90°]$, and the angle γ between

the corresponding solutions sets is

$$\gamma = \begin{cases} \Delta & \text{if } 0^\circ \le \Delta \le \arccos(1/4) \\ \arccos\left(\frac{-\sin 3\Delta}{3\sin\Delta}\right) & \text{if } \arccos(1/4) \le \Delta \le 90^\circ. \end{cases} \tag{4.8}$$

(Refer to [36].)

The rate of convergence depends upon the angle between the solution sets. If the angle is small, then the solution sets are similar, and the rate of convergence is slow. This can be seen from the two dimensional model depicted in Fig. 4.10, where large angles bring the iterate quickly closer to the intersection set.

It is therefore natural to expect that the rate of convergence can be improved if the projections are taken in a different order. To investigate this conjecture, and also to study the convergence rate, the iterated convergence procedure was tested on the experimental data with three different orderings of the projections. The base test used a spacing between projections of 105.1°, i.e., $\theta_1 = 0^\circ$, $\theta_2 = 105.1^\circ$, $\theta_3 = 210.3^\circ$, ... (Note that this spacing allows the entire set of 315 projections to be used.) Fig. 4.13 plots the distance of the iterates f_k from the final iterate (essentially f_∞), measured as the square root of the sum of the squares of the pointwise differences. Also shown on this plot are the results using the natural ordering as well as a between projection spacing of 90.3°. The decay of the relaxation parameter to 0 guarantees the eventual convergence of the iterates with any ordering, but note that the actual limit arrived at depends upon the ordering chosen. Note also that the ordinate variable marking the iterate count k

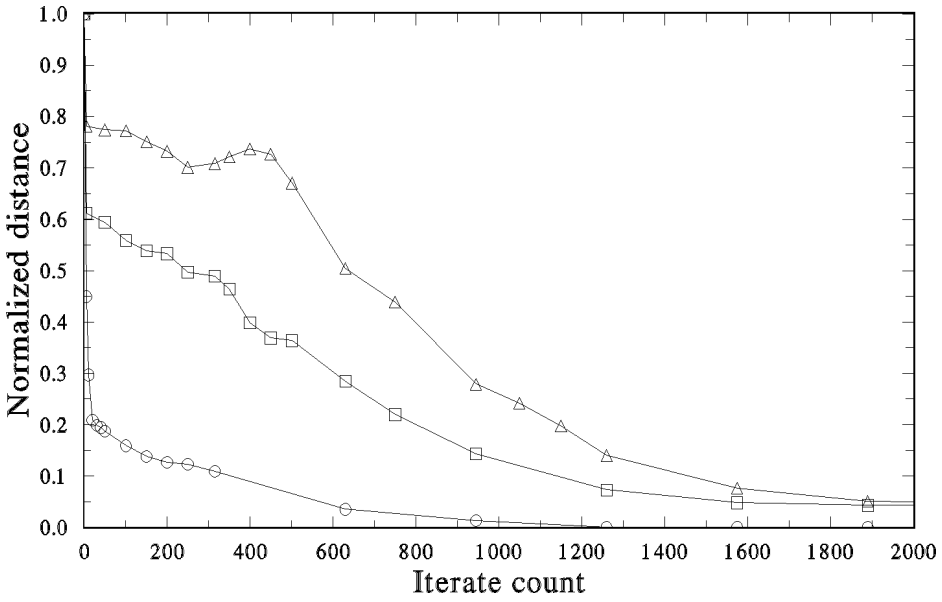


Figure 4.13: Comparison of the convergence rates of the iterated reconstruction method with different orderings of the projection data. △—1.14° increments, ⊙—105.1° increments, □—90.3° increments.

also effectively marks the size of the relaxation parameter $\lambda = 0.998^{k-1}$. Thus we see from Fig. 4.13 that the iterates using the natural ordering do not even begin to converge until the iterate count gets rather high, i.e., until the relaxation parameter becomes small. On the other hand, with the base ordering (105.1° spacing) the convergence begins from the first iterate.

The third curve in Fig. 4.13 corresponds to a spacing of 90.3°. Although the rate of convergence is better than with the natural ordering, it is not nearly as good as the convergence obtained using the base ordering of 105.1° spacing. According to Eq. 4.8, the angle between the solution sets is 70.5°, which is not substantially different from the angle between the subspaces using the base ordering (74.9°) or

much less than the theoretical maximum angle of 75.5°. (The base ordering angle of 74.9° is the closest to 75.5° available from the projection data of 315 evenly spaced directions.)

Note, however, that with the 90.3° spacing every second projection is nearly the same, e.g., the projection from 0° varies little (except for reflection) from the projection from 180.6°. So the iterates converge quickly to a function lying (roughly) in the intersection of the solutions sets for directions 0° and 90.3°. But the final convergence is towards a function lying in the intersection of *all* the solution sets—a much smaller set. In particular, the function toward which the early iterates converge is *not* the correct limit function. On the other hand, the base ordering of 105.1° provides projections around 360° in a much more evenly balanced fashion. The first 12 directions in the base ordering are 0°, 105.1°, 210.3°, 315.4°, 60.6°, 165.7°, 270.9°, 16°, 121.1°, 226.3°, 331.4°, and 76.6°. It is not until the thirteenth direction, 181.7°, that a close to repetition occurs. (The projection information from 181.7° will be very close to that from 0°.) So the early iterates converge towards the intersection of 12 solution sets from well spaced directions, as opposed to the effective 2 solutions set intersection towards which the iterates converge if a 90.3° spacing is used.

The final iterates do not contain large visual differences. The final iterate using the base ordering (181.7° steps) is shown in Fig. 4.14.

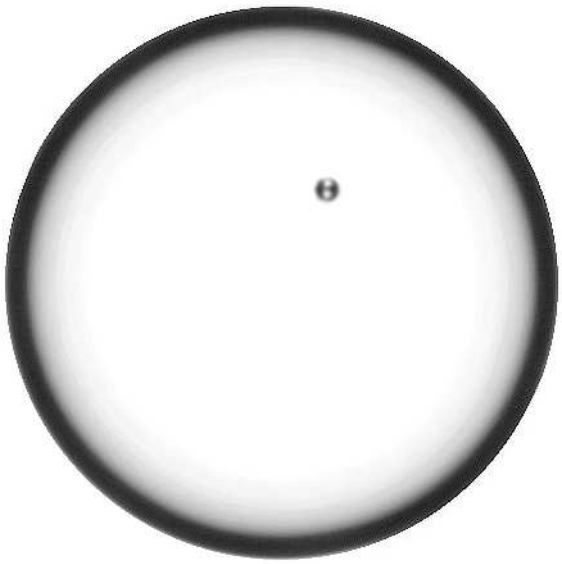


Figure 4.14: Iterated reconstruction method final iterate (f_{1260}) using complete experimental data set (315 projections).

4.3.4 A priori data

This work makes use of two types of a priori data. The next section discusses the use of allowed ranges for the reconstructed function, e.g., requiring all densities to be non-negative. This type of a priori information is needed for all reconstructions requiring a large number of iterations. The succeeding section discusses the use of a good initial "guess" to start the iteration. For example, if one is checking for flaws in a manufactured part then one could use an ideal unflawed part as the first iterate. This allows for reconstructions with very large missing ranges.

4.3.4.1 Limit ranges

Physical objects must have nonnegative densities, so it is reasonable to require the reconstructed function to have nonnegative densities as well. Similarly, it is often the case that some upper bound can be placed on the densities as well. (Or perhaps even several allowed ranges of densities.) Such restrictions can be incorporated into the iterative reconstruction method by adjusting intermediate iterates to conform to the restrictions. For example, each iterate f_k can be modified before being used to generate the next iterate f_{k+1}. One simple way to modify the an iterate f_k is to simply change any negative value to 0 and any value above a preset maximum to that maximum value.

On simulated data this type of adjustment accelerates the convergence by a considerable amount. Unfortunately, on the experimental data this procedure prevented convergence from occurring at all. However, instead of adjusting each iterate, the process can be mollified by adjusting the iterates at larger intervals. For example, adjust only f_{315}, f_{630}, f_{945}, etc.. It is well known that after some number of iterations (generally 3 or 4 times through the entire set of projections) that if the projections are not consistent, then the iterates will diverge and so the reconstruction quality deteriorates. (See, for example, [14, 2].) This effect was also observed in the reconstructions presented here. But in these examples, at least, the effect was caused by an accumulation of high density in some of the individual reconstruction pixels. (The total mass contained in each iterate is constant.) The

discrepancy of the behavior with the theory outlined previously is apparently due to the discrete nature of the implementation, as opposed to the continuous nature of the model discussed in theory.

This obstacle to large iterate reconstructions can be overcome, however, by using a priori range limitations as discussed above. This technique was employed on all reconstructions presented in this work that required more than two passes through the complete set of projection data, in particular it was used on the examples presented in Fig. 4.13 and 4.14. For example, the experimental data set consisted of 315 projections. In the iterative reconstructions, iterates f_{630}, f_{945}, ..., were modified by truncation to the range $[0, 0.016]$. The succeeding iterates (f_{631}, f_{946}, ...) were formed using the corresponding modified iterate.

4.3.4.2 Selection of the initial iterate

Another important factor in iterative reconstructions is the choice of the initial iterate. In all of the preceding examples the initial iterate was chosen to be the 0 function. Clearly, if the initial iterate is close to the final iterate then the convergence will be much quicker. So this then is a useful approach to using a priori information. But there is also a deeper significance. It can be shown [14] that the iteration converges to the unique element of the intersection of the solution set that is closest to the initial iterate. So the choice of initial iterate can affect the final reconstruction.

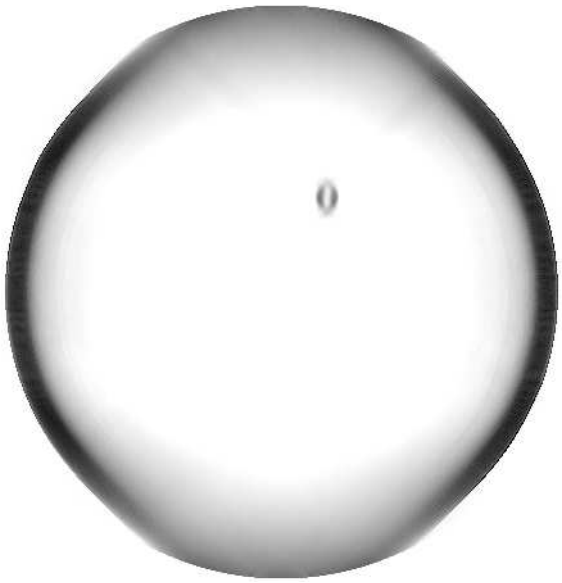


Figure 4.15: Iterated reconstruction from experimental data without projections from the 90° range about the vertical axis, using 0 as initial iterate.

This effect is most apparent when the intersection of the solution sets is large. (Of course, if this intersection consists of only one element then the initial iterate does not affect the limit iterate, although it will affect the rate of convergence.) Fig. 4.15 shows the iterated reconstruction of the experimental data without the projections from the 90° cone about the vertical axis, starting with the 0 function as the initial iterate. Compare this to Fig. 4.16 which shows the same reconstruction, but using as the initial iterate a homogeneous circular shell matching the outer shell of the reconstruction shown in Fig. 4.14. The normalized distance between Fig. 4.15 and Fig. 4.14 (as defined in Section 4.3.3 and used in Fig. 4.13) is 0.39, while the normalized distance between Fig. 4.16 and Fig. 4.14 is 0.27.

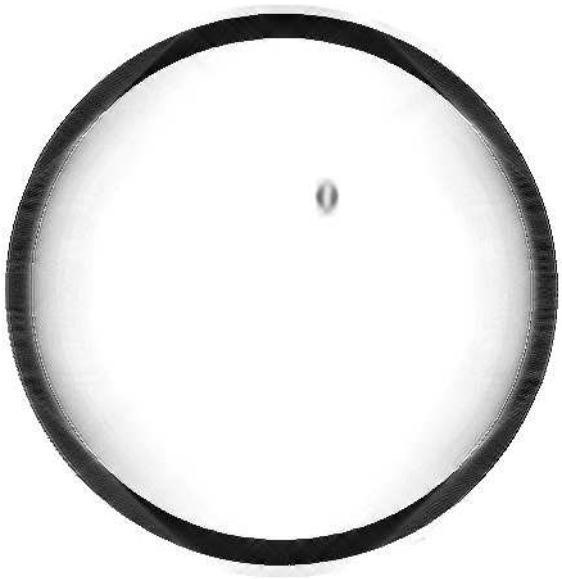


Figure 4.16: Iterated reconstruction from experimental data without projections from the 90° range about the vertical axis, using outer cylinder as initial iterate.

CHAPTER V

Summary to Part I

Part I of this dissertation studies and presents new results in computer aided tomography and feature detection and recognition. These are areas of increasing importance in nondestructive evaluation. To understand the images examined here it is useful to begin with a study of the formation of radiographic images.

Accordingly, a novel method is presented for the simulation of radiographs of 3 dimensional bodies. Not only can this be used to study the form of radiographs of complex objects, but it was needed to generate noise-free projections for the tomographic reconstruction algorithms studied later in the dissertation. The technique is based on the superposition of analytically calculated radiographs for several base element types such as spheres, ellipsoids, and rectangular solids. The algorithm was implemented, and the computational requirements were meager enough that the implementation was able to generate sample radiographs in a matter of minutes on an 386-based IBM PC. Although the technique is not as general as a ray-by-ray tracing method (which does not assume any special body geometry), it

is general enough for most industrial purposes as manufactured parts are generally built from standard geometric shapes. Moreover, the computational requirements of this algorithm are many times less than the more general ray tracing algorithm. As a further point, the base element nature of the presented algorithm should allow for a straightforward interface to design CAD packages. In this manner the nondestructive evaluation of manufactured objects could be intimately linked into the design process. It would be possible to introduce defects into a design and generate radiographs of the affected region. This allows the designer to ascertain what types of flaws will be detectable using radiography and allow adjustment to the design or direction to evaluation.

The first tomographic reconstruction technique studied in this dissertation is a 3D cone beam algorithm suggested by Feldkamp [3]. Although the algorithm is not new, and has been implemented elsewhere, we developed our own implementation of this algorithm and tested it on both simulated and experimental data. The implementation is nontrivial and provides useful background to the problems and limitations inherent in 3 dimensional tomographic reconstructions. When restricted to the 2 dimensional midplane, this algorithm reduces to the popular filtered backprojection algorithm, and so in this manner our implementation could also be used as a standard reconstruction technique against which other novel reconstruction approaches could be measured. In particular, a comparison is made between reconstruction with simulated versus experimental data.

Following this an in-depth study of limited angle tomography is undertaken. First a method of replacing the missing data via analytic continuation is undertaken. A Cauchy integral using data from the known directions is used to extrapolate the unknown data from the missing directions. The technique was suggested in a short mathematical paper by Palamdov and Denisjuk [4], though to the best of my knowledge this is the first actual implementation of this idea. Numerical instabilities make the algorithm difficult to implement, and a novel filter was introduced by this author to make the problem more tractable. In the end, however, the inherent instabilities prevented this technique from being successful if the missing angular range were larger than about 20°. Example results using this algorithm on experimental data with a 20° wedge removed are provided.

This analytic continuation method is built upon Fourier transform methods, and like all such approaches does not lend easily to the introduction of a priori data. So to use a priori information (for example reconstruction value range restrictions) we made use of iterative (algebraic) reconstruction techniques. The base algorithm used was an original implementation with modifications to the known SIRT reconstruction algorithm. With this implementation quantitative justification on experimental data is provided illustrating the known theoretical results that the natural ordering of the projection data is suboptimal. Studies on this result in the past have been of a qualitative nature or done on simulated data. Another known effect of the iterative reconstruction method on experimental data

is iterate divergence after 1 or 2 complete iterations through the available data. Presented in this dissertation is a new result showing that this divergence can be controlled by using a priori bounds on the reconstructed densities. Finally, an example is provided showing the effects of initial estimate selection on iterative reconstruction of experimental data. This information used to make this selection should also be considered a priori information.

A natural extension of the tomography work would be methods for automatic detection of flaws (or other features) in radiographs and tomographic reconstructions. Our research funding, however, was toward different but parallel problems, namely printed circuit board inspection and fingerprint identification. This is the topic discussed in Part II.

Part II

Computer recognition of plane images via feature encoding and matching

CHAPTER VI

Background to Part II

There is an increasing need in industry for automated control and inspection. Computer vision can play an important role in meeting these needs. Although pixel by pixel image comparison (subtraction) can be a useful technique in well controlled situations, general image recognition problems require more powerful feature based techniques [37, 38, 39, 40]. This dissertation presents applications of feature based methods to defect detection and fingerprint identification.

6.1 Printed circuit boards

There is an ever increasing need in industry for automatic inspection of printed circuit boards (PCB's). The chief requirements for such systems is high speed, adaptability, and precision. Current inspection systems utilize design rule and/or reference comparison methods [38, 41].

Design rule methods attempt to detect deviations in the board under test from design specifications. The most notable such specification is wire width. Any metal

trace that falls below a certain width may be unable to carry the required amount of current without unacceptable voltage drops, and is likely to indicate a trace break and possible board underetching on the whole. Similarly, a too wide metal trace is indicative of an overetched board and suggests the existence of inadvertent short circuits. See [42, 43, 44, 45]. These methods typically use morphological transformations such as dilation and eroding to detect design rule violations. Such methods are memory and computationally expensive, or else require specialized hardware. Moreover, certain types of defects, such as missing or additional traces, cannot be detected by design rule methods.

Reference comparison methods compare the board under test against a reference (defect-free) board. Such comparisons can be made via pixel by pixel subtraction [46, 47], but this requires precise board alignment and tight manufacturing tolerances. Feature based approaches are more robust and flexible [48, 43, 45]. These methods generally use morphological transformation such as eroding to detect features, which are then collected into a graph representing circuit elements and connectivity. The feature graph is then compared against a reference graph, and any changes denote defects.

Chapter VIII presents a PCB feature extraction method published in *Computer Vision, Graphics, and Image Processing* [1] that I co-authored with Alan Sprague. The method presented there included both a design rule section for wire width violation detection and a reference comparison section for detection of other types

of errors. It differs from previous methods chiefly by the absence of morphological transformations, thereby reducing the computational requirements. Only the reference comparison portion is presented in this dissertation.

6.2 Fingerprint identification

Fingerprint identification has a history dating back 100 years as a reliable method for identity verification [49]. Although fingerprint patterns (whorls, loops, and arches) are used for classification, actual identification is done at the minutia level using ridge endings and bifurcations. Many people share the same patterns on their fingers, but no two, not even identical twins, share the same distribution of minutiae. Although single fingerprints will generally contain over 50 minutiae (thumbprints often have over 100), identity can be established by matching only a small fraction of them. In fact, the courts in this country have held that only 15 such points are needed to legally establish identity. This is a conservative estimate—depending on the particular distribution involved only 6 or 7 may be required.

There are two facets to fingerprint identification. The simpler problem is to match two sets of 10-print inked fingerprint cards. Pattern classification can be used to whittle the comparison set (of perhaps millions of prints) down to a handful, which can be compared at the minutia level by hand. The more difficult task is to identify a lone, poor quality, latent print lifted from the scene of a crime.

This task has traditionally been performed by human hands and magnifying glass, comparing the prints of a few suspects against the latent. It is not possible without automation to compare a latent print against more than a small number of the inked prints the police may have on file. There are a number of factors that make automatic latent fingerprint identification a difficult task. The latent print generally contains only a small subset of the total number of minutiae, and its orientation (relative position and angle of rotation) is often unknown. The elasticity of human skin allows morphological variations to be exhibited between separate registrations of the same finger. Moreover, the latent print is generally of very poor quality, exhibiting false and missing minutiae due to dirt and smears.

Although there are now commercial systems to perform automatic fingerprint identification, they are expensive, proprietary, and generally perform poorly with regards to latent identification. Chapter IX and Section 10.2 present a method for fingerprint minutiae extraction and latent identification that was developed by this author with others under the direction of Prof. Rokhlin at The Ohio State University. It has been implemented on a IBM PC-based platform and has shown excellent promise. Included in the discussion are the results of latent matching against our library file of 700 prints.

Of course, often a feature based method of image analysis can be aided by judicious pre-processing and analysis of the digital image. The next chapter describes a novel approach for the extraction of edge direction and curvature information,

and applications to orientation specific filtering. This material stands on its own right, but was used especially in the fingerprint identification algorithm developed in later chapters.

CHAPTER VII

Using level curves in image analysis

This chapter details techniques for the extraction of direction and curvature information from general images. This work has been accepted for publication in *Computer Vision, Graphics, and Image Processing.*

7.1 Introduction

An image may be viewed as a surface $h(x, y)$ over the xy-plane, where peaks in the surface correspond to light areas of the image and valleys correspond to dark areas. The topological properties of this surface are useful in many applications [50, 51]. In particular, the level curves contain information that can be used in adaptive processing of the image for enhancement and pattern recognition. Tangents to the level curves run parallel to ridges in the image, so the tangent directions can be used for edge detection and enhancement. For example, obtaining this directional information is a necessary step in the processing of fingerprints for automated identification [52, 53] and for orientation specific filtering [54].

There are different approaches to extracting directional information. One common technique is to use an ideal edge as a template to detect the presence of an edge and to determine its orientation. Hueckel [55] and O'Gorman [56] used a modified form of this approach with least-squares minimization to match against an edge of arbitrary orientation. Kawagoe and Tojo [52] used a different method in their work with digitized fingerprints. On each 2×2 pixel neighborhood they made a straight comparison against 4 edge templates to extract a crude directional estimate, which was then arithmetically averaged over a larger region to obtain a more accurate estimate. Our approach uses a gradient-type operator to extract a directional estimate from each 2×2 pixel neighborhood, which is then averaged over a larger region by least-squares minimization to control noise.

Curvature of level curves is sensitive to broader topological properties of the image that can be used for feature detection and identification. For example, curvature is large near corners and peaks in the image, which are typical feature points used for pattern recognition [39]. Most previous work on curvature determination ([57], for instance) has centered on finding the curvature of a single parameterized curve that has been previously extracted from the image by a separate algorithm (a chain code algorithm, for example). Adjacent level curves, on the other hand, locally form a family of parallel curves. Our algorithm extracts curvature using data determined by these curves. This is accomplished without curve parameterization by fitting a family of concentric circles to the extracted tangent directions

via least-squares minimization.

The work [58] of Parent and Zucker is along similar lines. (See also [59].) They also extract tangent directions and curvature directly from the image without curve parameterization. In their work the tangent directions are quantitized to represent finite ranges. At each point in the image and for each of the allowed directions a probability is assigned. This probability is determined from two components. The first component is the result of convolutions with local "line detector" operators. The second component is a curvature compatibility factor, which measures how closely the local tangent directions model a smooth flow. The tangent directions are adjusted using an iterative relaxation approach. This algorithm goes further by using the tangent and curvature information to essentially extract curve traces. It is actually an integrated curve extraction algorithm.

The algorithm of Parent and Zucker assumes the existence of curves in the image and it is optimized towards the extraction of these curves. The algorithm we present does not assume the existence of curves and does not aim to extract them. It extracts only the tangent directions and curvature for further processing by other algorithms. It will work on any image possessing smooth level curves. Moreover, our algorithm directly extracts the directional information using least-squares minimization—no iterative processing is performed. Thus our algorithm is considerably less complicated and should be much faster than the algorithm of Parent and Zucker. On the other hand, the algorithm of Parent and Zucker ex-

plicitly handles curve intersections, an event our algorithm cannot interpret since it does not deal with curve traces. However, our algorithm will find curve intersections to have large curvature, so such events can be managed during subsequent processing.

In addition to the presentation of algorithms for the extraction of tangents and curvatures from level curves in digitized images, we show results of controlled tests of our implementation of these algorithms on computer generated data corrupted by simulated noise. Example applications are also provided showing the use of the tangent and curvature information in the image processing of fingerprints, materials microstructures, printed circuit boards, and radiographs.

7.2 Problem statement

Let us first introduce some terminology. Fig. 7.1 shows three level curves with tangent and curvature vectors marked. Let P be a point in the xy-plane as indicated and consider the level curve passing through that point. Let $\vec{r}(s)$ be a parameterization by arc length of this level curve, with $\vec{r}(0) = P$. The derivative $\vec{r}\,'(0)$ is the **tangent vector** to the level curve at point P. We are interested only in the unoriented direction of this vector, since both its magnitude and sign (+ or -) depend upon the parameterization. Choosing arc length parameterization forces the magnitude of $\vec{r}\,'(t)$ to 1, which makes the second derivative, $\vec{r}\,''(0)$, perpendicular to the tangent vector with magnitude proportional to the rate of change in

direction of the tangent vector. The vector $\vec{r}''(0)$ is called the **curvature vector** for the level curve at the point P. The direction of $\vec{r}''(0)$ is towards the center of the curvature of the level curve $\vec{r}(s)$, and the magnitude of $\vec{r}''(0)$, called the **curvature** of the level curve, is equal to the reciprocal of the curvature radius. This is evident in Fig. 7.1 where the magnitudes of the illustrated curvature vectors are largest on the inner level curve and smallest on the outer level curve.

We have so far assumed that the level set at the point P is one dimensional. However, if P corresponds to a local extrema or plateau, then the level set will be 0 or 2 dimensional, respectively. In this case it makes no sense to talk of a "level curve", and accordingly the tangent and curvature are not defined. The algorithms described in this chapter detect such occurrences.

In a digitized image the image values are only known on a discrete set of points (the sample nodes), so the curve $\vec{r}(s)$ and its derivatives can only be approximated. Rather than attempting to approximate $\vec{r}'(s)$ and $\vec{r}''(s)$ at the point P, we find an averaged value over a rectangular window around P using least-squares minimization. The effects of random noise are minimized in the average, so this method is noise tolerant. The choice of window size is application specific, depending on the size of the features of interest (with respect to the pixel size) and the noise level. There is generally a tradeoff between noise tolerance (large windows) and precision (small windows).

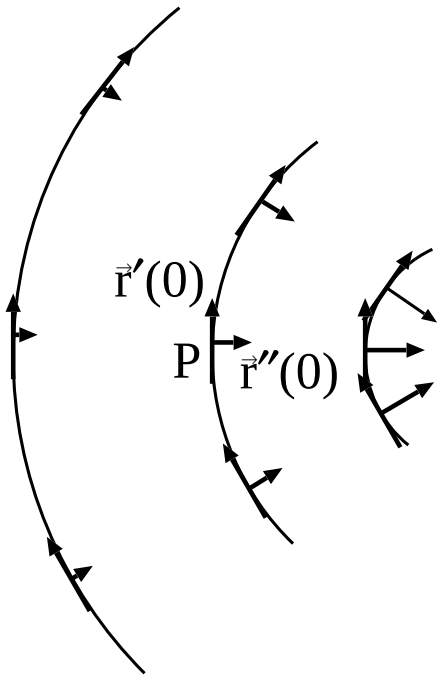


Figure 7.1: Schematic showing three level curves with associated tangent and curvature vectors.

7.3 Tangent vector calculation

7.3.1 Method outline

At each point in the averaging window we calculate a vector $\vec{n}$ that is normal to the surface $z = h(x, y)$. The tangent vector to the level curve at this point (if defined) lies in the xy-plane and is perpendicular to the normal. The averaged tangent vector for an averaging window is that (unit) vector in the xy-plane which comes closest to being perpendicular to all the surface normals in the window. We could average the tangents directly, but there are advantages to working with the surface normals. Consider, for example, the edge function defined by

$$g(x, y) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0, \end{cases}$$

which is a unit step with edge along the x-axis. Let $h(x, y)$ be a smooth approximation to $g(x, y)$. Fig. 7.2 is an illustration of this surface and several of the surface normals. Away from the x-axis the normals to $h(x, y)$ are parallel to the z-axis, and accordingly the tangents are undefined. As one moves near the x-axis, the normal vectors tilt away from the z-axis. If the approximation of $h(x, y)$ to the step function $g(x, y)$ is very tight, then along the x-axis the normal vectors to the surface $h(x, y)$ will be nearly perpendicular to the z-axis. In particular, the steeper the surface the smaller the angle between the surface normal and the xy-plane. Since noise corrupts shallow edges more easily than steep edges, we want to weight normals from steep edges more heavily in the average than normals from

shallow edges. Thus there are two advantages in working with the normal vectors: the normal vectors are always defined, and they carry information that is used to weight the average.

Of course, in a digitized image the function $h(x, y)$ is not known at every point in the xy-plane. Rather the value is known (with limited precision) on a finite grid of points, called the **sample nodes**. To simplify our discussion we shall assume that these nodes are evenly spaced on a square mesh, although the algorithm is easily adapted to other geometries. We call a set of 4 nodes that are corners of a grid-minimal square a **2x2 neighborhood**.

Our approach to calculating the tangent vector in a digitized image consists of two steps. First we estimate the surface normal at the center of each 2x2 neighborhood with the normal to a plane fitted through the image values at the 4 sample nodes. We refer to this as **point normal determination**. These normals are grouped into (possibly overlapping) regions called **tangent windows**. The second step consists of finding, for each tangent window, a vector $\vec{u}$ lying in the xy-plane that is nearly perpendicular to all the normals in that window. This vector is called the **averaged tangent vector**.

7.3.2 Point normal determination

Let (x_1, y_1), (x_2, y_2), ..., (x_m, y_m) be a collection of m sample nodes in the xy-plane, and let a_1, a_2, ..., a_m be the image values at these nodes. We fit to these

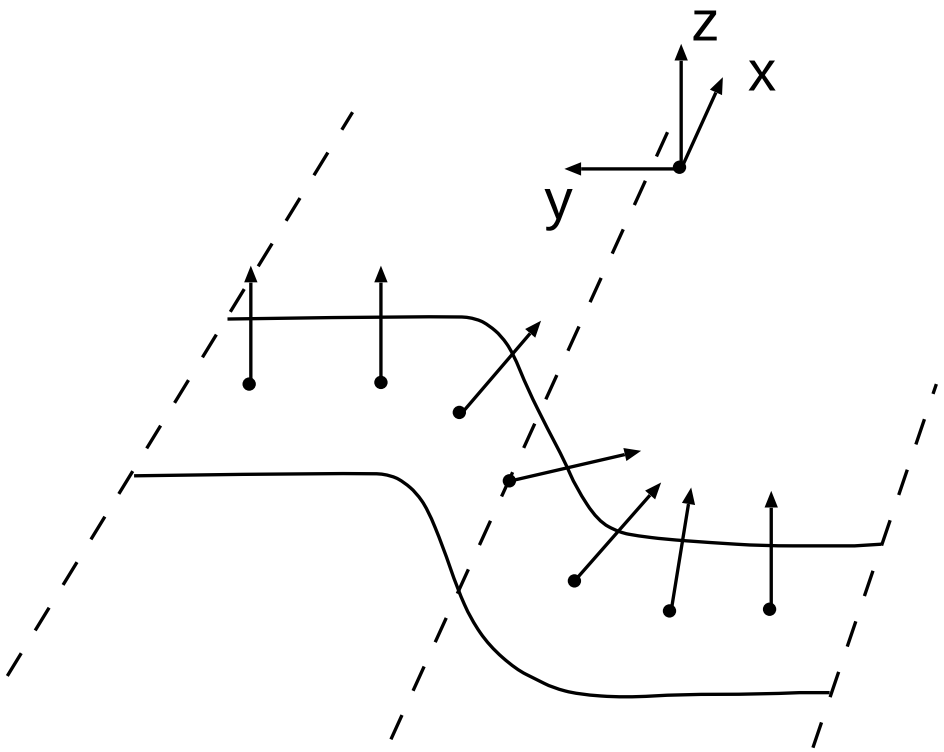


Figure 7.2: Illustration of a smooth approximation to a unit step function with edge along the x-axis. Also illustrated are several normal vectors to this surface.

values a plane $p(x,y)$ of the form $p(x,y) = -n_1x - n_2y + c$, which has normal vector $\vec{n} = (n_1, n_2, 1)$. (The digitized values are restricted to a finite range, ensuring that $\vec{n}$ will not be perpendicular to the z-axis.) We want to select values for $n_1,\ n_2,$ and c to minimize the sum of the squared difference of the image values with the plane values over the m sample nodes. That is,

$$\min_{n_1,n_2,c} \sum_{\substack{\text{sample}\\ \text{nodes}}} |h(x,y) - p(x,y)|^2 = \min_{n_1,n_2,c} \left\| \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} - \begin{pmatrix} -x_1 & -y_1 & 1 \\ -x_2 & -y_2 & 1 \\ \vdots & \vdots & \vdots \\ -x_m & -y_m & 1 \end{pmatrix} \begin{pmatrix} n_1 \\ n_2 \\ c \end{pmatrix} \right\|^2, \tag{7.1}$$

which is a standard least-squares minimization problem (see [60]). Let us set $\vec{w}^1 = (-x_1, -x_2, \ldots, -x_m)^T$, $\vec{w}^2 = (-y_1, -y_2, \ldots, -y_m)^T$, and $\vec{w}^3 = (1, 1, \ldots, 1)^T$. Then this minimization problem can be viewed as finding that (m-dimensional) vector in the span of $\{\vec{w}^1, \vec{w}^2, \vec{w}^3\}$ that is closest to the vector $\vec{a} = (a_1, a_2, \ldots, a_m)^T$. This is given simply by the orthogonal projection of $\vec{a}$ onto the subspace spanned by $\{\vec{w}^1, \vec{w}^2, \vec{w}^3\}$.

For example, consider the 2x2 neighborhood depicted in Fig. 7.3. Here $m = 4$, and $\vec{w}^1 = (-1, 1, 1, -1)^T,\ \vec{w}^2 = (-1, -1, 1, 1)^T,$ and $\vec{w}^3 = (1, 1, 1, 1)^T$. Since the $\vec{w}^i$'s are orthogonal and $\|\vec{w}^i\|^2 = 4$ for each $i = 1, 2, 3$, it follows that the minimizing

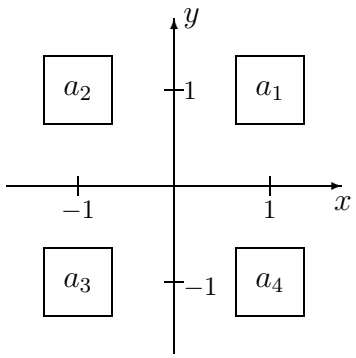


Figure 7.3: Local coordinate system imposed onto each 2x2 neighborhood.

vector is

$$\frac{\langle \vec{a}, \vec{w}^1 \rangle \vec{w}^1}{4} + \frac{\langle \vec{a}, \vec{w}^2 \rangle \vec{w}^2}{4} + \frac{\langle \vec{a}, \vec{w}^3 \rangle \vec{w}^3}{4} =$$
$$\frac{-a_1 + a_2 + a_3 - a_4}{4}\vec{w}^1 + \frac{-a_1 - a_2 + a_3 + a_4}{4}\vec{w}^2 + \frac{a_1 + a_2 + a_3 + a_4}{4}\vec{w}^3$$

where $\langle \cdot, \cdot \rangle$ denotes the usual scalar product.

Comparing to Eq. 7.1, we see that the minimizing values are

$$\begin{aligned} n_1 &= \frac{-a_1 + a_2 + a_3 - a_4}{4} \\ n_2 &= \frac{-a_1 - a_2 + a_3 + a_4}{4} \\ c &= \frac{a_1 + a_2 + a_3 + a_4}{4}. \end{aligned} \tag{7.2}$$

Therefore we approximate the surface normal at the center of the 2x2 neighborhood by $\vec{n} = ((-a_1 + a_2 + a_3 - a_4)/4, (-a_1 - a_2 + a_3 + a_4)/4, 1))^T$ (the value of c is irrelevant). Note that the magnitude of this vector depends on the angle it makes with the xy-plane. If the normal is parallel to the z-axis then the magni-

tude is minimized to 1, and the magnitude grows as the angle with the xy-plane is decreased. This variation is used to weight the average discussed in the next section.

Of course, one can choose a neighborhood larger than 2x2 (see [61]). The advantages of a larger neighborhood are increased directional accuracy and noise tolerance. Noise tolerance is not an issue here, since we do an independent averaging step to control noise (Section 7.3.3). Directional accuracy may be a factor, however, depending on the type of edges examined. It is well known that gradient type operators such as this one have orientation biases due to discretization. A thorough study of this effect for step edges was done by Kitchen and Malin [62]. (Refer also to O'Gorman [56], which proposes the use of Walsh functions for edge detection.) Nonetheless, we found this effect to be insignificant in our work, and for the smooth edges studied in Section 7.3.5 the orientation bias was found to be less that 3°. Moreover, larger neighborhoods increase the computational requirement and can have problems relating to edge curvature [59].

7.3.3 Determining the averaged tangent direction

We want next to find a vector (the unit tangent vector) that is nearly perpendicular to the collection of normal vectors $\{\vec{n}^k\}$, where the index k runs over all 2x2 neighborhoods in the tangent window. One approach is to calculate the tangent direction to each normal, express that direction as an angle between say 0° and

180°, and then calculate the arithmetic average of these directions. This is essentially the method used by Kawagoe and Tojo [52]. There are some difficulties with this procedure, however. For example, notice that the average of the directions 0° and 178° should be 179° (not 89°). (See [63] for background on managing directional data.) We take instead a least-squares minimization approach. First we discard the z-components (which are identically 1) to get the collection $\{\vec{v}^k\}$, where $\vec{v}^k = (n_1^k, n_2^k)^T$. Note that the magnitude of the vector $\vec{v}^k$ is given by the cotangent of the angle between the normal vector $\vec{n}^k$ and the xy-plane. Let $\vec{u} = (u_1, u_2)^T$ be the unit tangent vector that we want to determine. Formally,

$$\begin{aligned} \text{minimize} \quad & \sum_k |\langle \vec{v}^k, \vec{u} \rangle|^2 \\ \text{subject to} \quad & \|\vec{u}\| = 1. \end{aligned}$$

(See [60].)

Let $F(u_1, u_2)$ be the minimization function, i.e., $F(u_1, u_2) = \sum_k |\langle \vec{v}^k, \vec{u} \rangle|^2$. Let $A = \sum_k (v_1^k)^2$, $B = \sum_k (v_2^k)^2$, and $C = \sum_k v_1^k v_2^k$. Then

$$F(u_1, u_2) = (u_1, u_2) \begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \tag{7.3}$$

Thus F is the quadratic form associated with the (real) symmetric matrix

$$S = \begin{pmatrix} A & C \\ C & B \end{pmatrix}.$$

The eigenvalues for this matrix are the extremal values of the function F for $\|\vec{u}\| = 1$. Let us order the eigenvalues so that $\lambda_1 \geq \lambda_2$. Then the minimum value

for F is λ_2, and the corresponding eigenvector is the solution to the minimization problem, i.e., yields the averaged tangent direction. (If $\lambda_1 = \lambda_2$ then we have the degenerate case where there is no preferred direction. This indicates a peak or plateau in the image, or else a region that is overcome by random noise.)

The eigenvalues for S are

$$\begin{aligned} \lambda_1 &= \frac{1}{2}\left((A+B) + \sqrt{(A-B)^2 + 4C^2}\right) \\ \lambda_2 &= \frac{1}{2}\left((A+B) - \sqrt{(A-B)^2 + 4C^2}\right). \end{aligned} \tag{7.4}$$

If $C = 0$ then the original matrix S is diagonal and the eigenvector corresponding to λ_2 is either $(1,0)^T$ or $(0,1)^T$, depending on whether $A < B$ or $B < A$. Otherwise one has the relation

$$\begin{aligned} u_2 &= u_1(\lambda_2 - A)/C \\ &= u_1\left(\frac{B-A}{2C} - \sqrt{\left(\frac{B-A}{2C}\right)^2 + 1}\right), \end{aligned}$$

which defines the averaged tangent direction.

7.3.4 Region contrast and consistency

In addition to determining the tangent direction, the calculations in the preceding sections also give measures of region contrast and tangent consistency. For example, if the contrast in a region is large, then many of the normal vectors calculated in Section 7.3.2 will lie close to the xy-plane, and the projected vectors $\vec{v}^k$ (from Section 7.3.3) will be large. Conversely, if the contrast is small, then most of the

normal vectors will be nearly parallel to the z-axis, and the vectors $\vec{v}^k$ will be small. Thus the region contrast is gauged by

$$\begin{aligned} C_R &= \sum_k \left\| \vec{v}^k \right\|^2 \\ &= \sum_k (n_1^k)^2 + (n_2^k)^2 = A + B, \end{aligned}$$

where A and B are defined in Section 7.3.3. Normalization of C_R requires knowledge of the maximum possible magnitude of the normal vectors (which depends on digitization restrictions) in addition to the size of the averaging window (i.e., the number of vectors in the summation). For example, in our system the digitized values lie between 0 and 255, and suppose we choose a 9×9 averaging window. The maximum value for any $\|\vec{v}^k\|$ is 255/2 (refer to Eq. 7.2), and there are $8 \times 8 = 64$ 2x2 neighborhoods in the averaging region, so the maximum possible value for C_R is $C_{\max} = (255/2)^2 \times 64$. The normalized contrast score is defined by $C_N = C_R / C_{\max}$.

Tangent consistency is measured by the extent to which the averaged tangent direction $\vec{u}$ is perpendicular to the collection $\vec{n}^k$. This provides an estimate of the noise level in the averaging window. Let E_R denote the sum of the squared error,

$$E_R = \sum_k |\langle \vec{n}^k, \vec{u} \rangle|^2 = \sum_k |\langle \vec{v}^k, \vec{u} \rangle|^2,$$

which is of course just the function F of Section 7.3.3 evaluated in the averaged tangent direction. This minimal value of F is given in Eq. 7.4, i.e., $E_R = \lambda_2$. This value can be normalized by dividing by C_R. A value of zero indicates a perfect fit

(all the vectors $\vec{n}^k$ are perpendicular to the averaged tangent direction), whereas $E_R/C_R = 1$ indicates that all the vectors $\vec{n}^k$ are parallel to the averaged tangent direction—the worst case. Of course the latter does not occur since in this case selecting the perpendicular direction as the tangent direction gives a perfect fit. In fact, the minimization ensures that $E_R/C_R \leq 1/2$. Therefore, the normalized consistency error is defined to be $E_N = 2E_R/C_R$.

Combined, the two parameters C_N and E_N give a measure of the reliability of the calculated tangent direction. Ideally the region will be high contrast (large C_N) with high consistency (small E_N).

7.3.5 Controlled tests

To test the stability of the tangent calculation and the usefulness of the consistency score E_N, we performed experiments with two idealized surfaces (a smooth edge and a sinusoid) corrupted by simulated noise. The results are displayed in Tables 7.1 and 7.2. Part (a) in each table presents the results using a small (9×9) tangent window, while part (b) shows the results using a larger (19×19) tangent window. The data in Table 7.1 (a) and Table 7.2 (b) are represented graphically in Fig. 7.4 and Fig. 7.5, respectively. Graphs of the data in Table 7.1 (b) and Table 7.2 (a) are very similar to Fig. 7.4, and so are not reproduced here. The reader may examine the tabular data for details.

The first surface for which we performed tests was a smooth edge. The profile

Table 7.1: Experimental results of tangent calculation for a smoothed edge passing through the center of the tangent window at an angle of 40°. Results in (a) are from a 9×9 window, (b) from a 19×19 window. Noise was introduced by adding a sequence of computer generated zero-mean uncorrelated Gaussian random variables.

	Angle		E_N		C_N	
SNR	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
No noise	39.6°	–	0.00	–	0.15	–
100	39.6°	0.57°	0.06	0.01	0.16	0.01
50	39.6°	0.90°	0.11	0.02	0.17	0.01
25	39.6°	1.5°	0.20	0.04	0.19	0.01
10	39.6°	3.4°	0.38	0.06	0.24	0.02
5	39.7°	6.3°	0.54	0.08	0.34	0.04

(a)

	Angle		E_N		C_N	
SNR	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
No noise	39.4°	–	0.00	–	0.07	–
100	39.4°	0.41°	0.14	0.01	0.08	0.00
50	39.4°	0.75°	0.23	0.02	0.09	0.00
25	39.4°	1.5°	0.38	0.03	0.11	0.01
10	39.4°	3.5°	0.61	0.04	0.17	0.01
5	39.6°	6.8°	0.75	0.04	0.28	0.02

(b)

Table 7.2: Experimental results of tangent calculation for a sinusoidal wave passing through the center of the tangent window at an angle of 40°. Results in (a) are from a 9×9 window, (b) from a 19×19 window. Noise was introduced by adding a sequence of computer generated zero-mean uncorrelated Gaussian random variables.

	Angle		E_N		C_N	
SNR	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
No noise	39.7°	–	0.00	–	0.11	–
100	39.8°	0.63°	0.05	0.01	0.11	0.00
50	39.8°	0.95°	0.09	0.02	0.12	0.00
25	39.8°	1.5°	0.17	0.03	0.13	0.01
10	39.8°	3.1°	0.34	0.06	0.16	0.01
5	39.9°	5.6°	0.50	0.07	0.22	0.02

(a)

	Angle		E_N		C_N	
SNR	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
No noise	39.9°	–	0.00	–	0.11	–
100	39.8°	0.21°	0.05	0.00	0.11	0.00
50	39.8°	0.34°	0.10	0.01	0.12	0.00
25	39.8°	0.59°	0.18	0.01	0.13	0.00
10	39.8°	1.3°	0.35	0.03	0.16	0.01
5	39.9°	2.5°	0.52	0.03	0.22	0.01
2	39.9°	6.0°	0.72	0.04	0.39	0.03
1	40.5°	11°	0.83	0.04	0.68	0.05

(b)

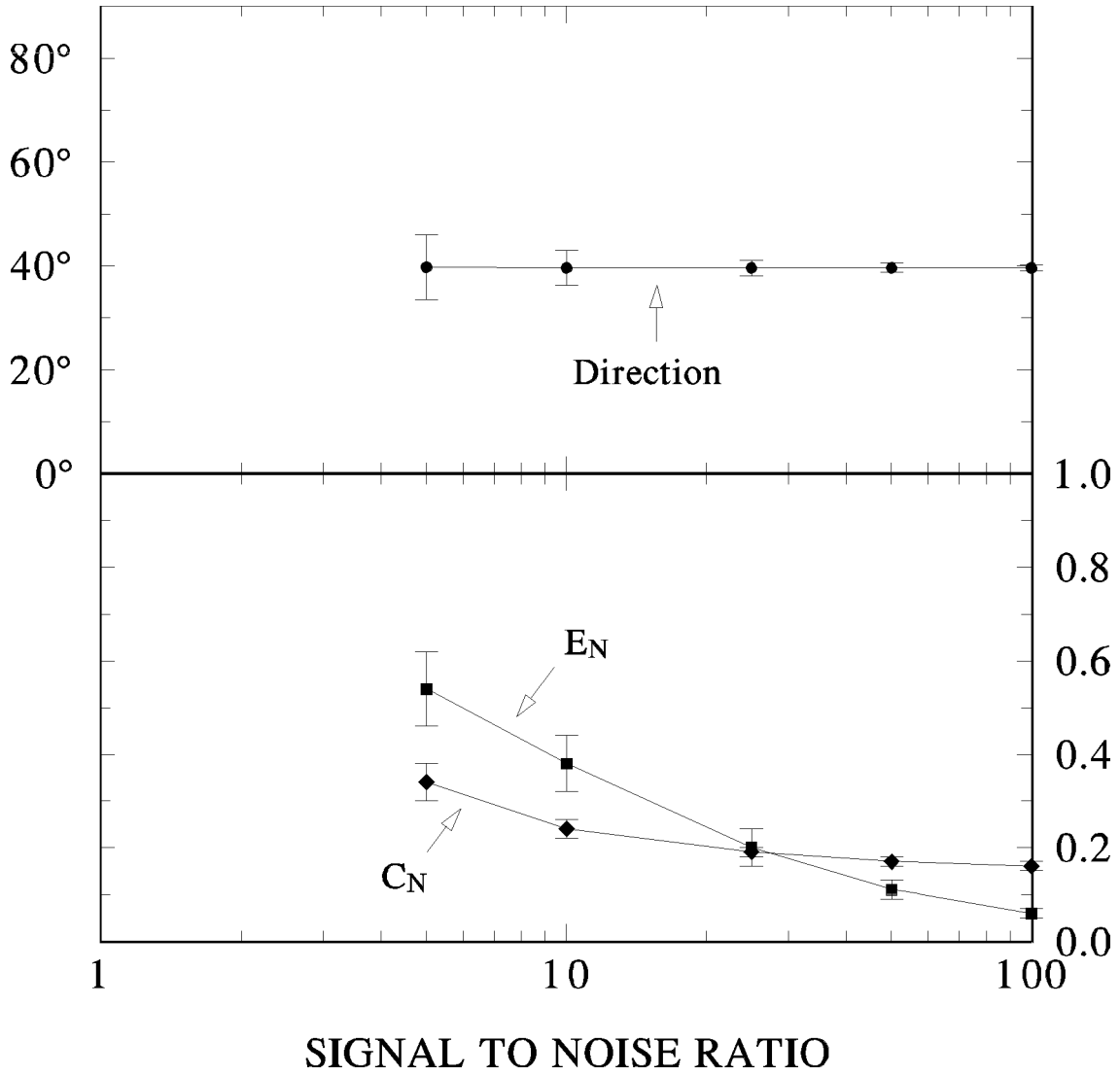


Figure 7.4: Graph of results of tangent calculations for a smoothed edge (see Fig. 7.6) passing through the center of a 9×9 tangent window at an angle of 40°. Noise was introduced by adding a sequence of computer generated zero-mean uncorrelated Gaussian random variables. The tangent direction (in degrees), normalized error (E_N), and normalized contrast score (C_N) are displayed as functions of the signal to noise ratio. The error bars mark one standard deviation.

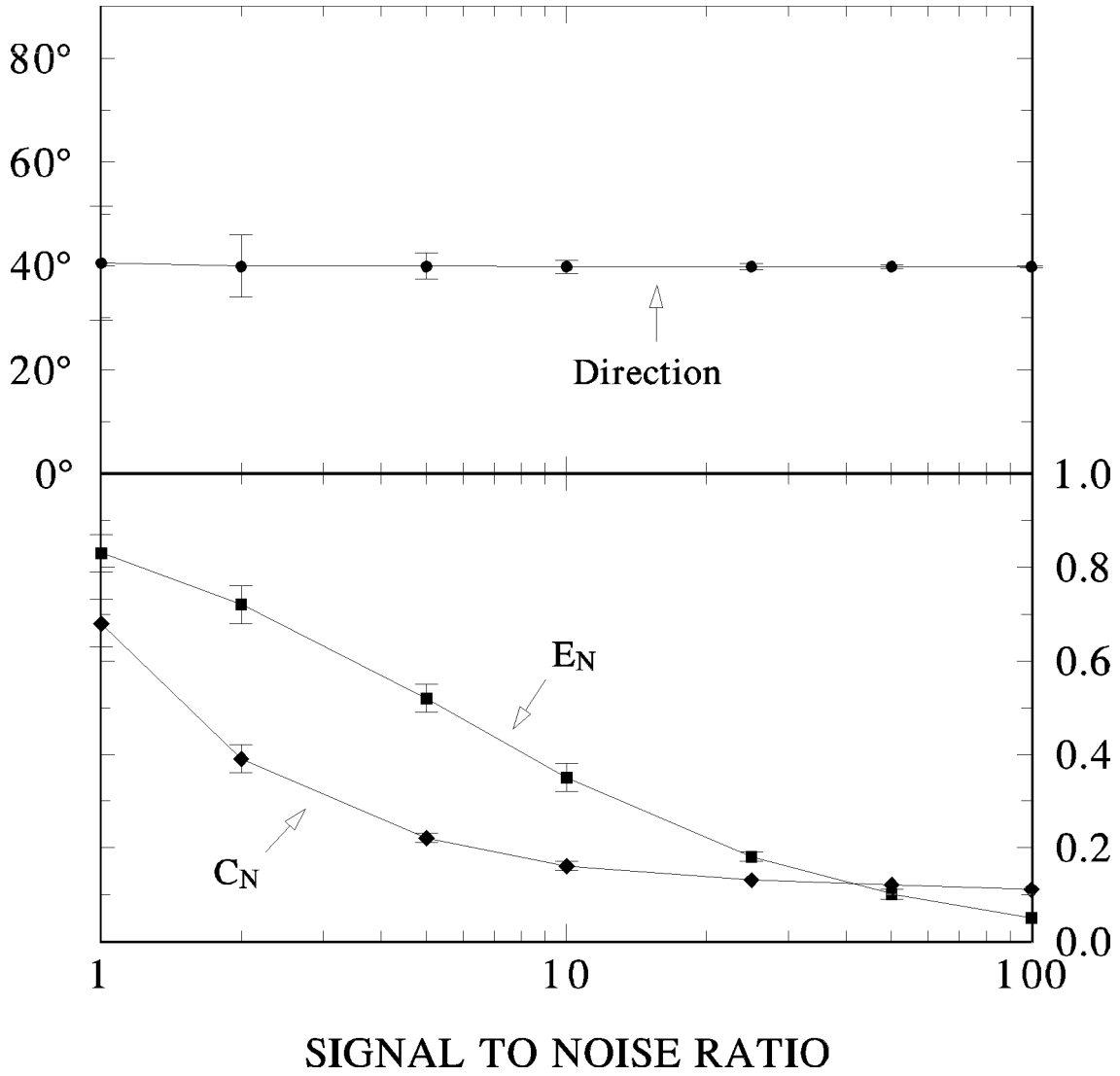


Figure 7.5: Graph showing results of tangent calculations for a sinusoid (see Eq. 7.5) passing through the center of a 19×19 tangent window at an angle of 40°. Noise was introduced by adding a sequence of computer generated zero-mean uncorrelated Gaussian random variables. The tangent direction (in degrees), normalized error (E_N), and normalized contrast score (C_N) are displayed as functions of the signal to noise ratio. The error bars mark one standard deviation.

80	80	80	80	80	80	81	84	95
80	80	80	80	80	81	85	100	150
80	80	80	80	81	86	107	157	172
80	80	80	82	88	116	161	173	175
80	81	82	91	128	165	174	175	176
81	83	95	140	168	174	176	176	176
84	99	149	170	175	176	176	176	176
106	156	171	175	176	176	176	176	176
161	172	175	176	176	176	176	176	176

Figure 7.6: Tangent window pixel values for a smooth edge passing through the center of the window at an angle of 40°.

for this edge was defined by

$$h(t) = \begin{cases} 128 + 48(1 - e^{-2.3t}) & \text{if } t \geq 0 \\ 128 - 48(1 - e^{2.3t}) & \text{if } t < 0 \end{cases}$$

where t is the distance (in pixels) from the center of the edge. This profile has an edge transition range, measured from the 10% down points, of two pixels. We used this profile to generate a smooth step (similar to that depicted in Fig. 7.2) through the center of a tangent window at an angle of 40°. The resulting pixel values, after the required rounding to integer, are shown in Fig. 7.6. Noise was added to this in the form of a sequence of computer generated uncorrelated zero-mean Gaussian random variables.

Table 7.1 (a) shows statistics for the tangent calculation using a 9×9 tangent window. (The window dimensions are chosen odd so that the center of the window will correspond to a pixel sampling point.) The first column lists the signal-to-noise ratio (SNR), defined here as the ratio of the variance of the original (no noise) image to the variance of the added noise. For each SNR level, 10^4 trials were

performed, and the mean and standard deviation of the tangent angle, consistency score, and contrast score were collected. Notice that even in the noise-free case, the angle calculation is 0.4° off from the true value of 40°. This is due to the discretization of the edge and because the point normal calculation works under the assumption that a plane is a good local approximation to the surface. The magnitude of the error depends upon the angle and the shape of the edge. As shown in [62], this error can be significant across a step edge. The effect is less noticeable across a smooth edge. In fact, for the smooth edge described above, the calculated angle differs from the true angle by less than 3° regardless of the edge orientation. Nonetheless, one could improve the directional accuracy by improving the point normal calculations (Section 7.3.2), either by increasing the size of the local neighborhood or by using a higher order surface (instead of a plane), albeit at the cost of increased computation time. The contrast scores are not affected by the orientation of the edge, and the effect of edge orientation on the consistency (error) scores is less than 10%.

The data from Table 7.1 (a) are displayed graphically in Fig. 7.4, where the error bars denote one standard deviation from the mean. (The distributions about the mean are not quite Gaussian, but are close enough so that roughly 68% of the trials fell within one standard deviation of the mean, and approximately 95% fell within two standard deviations.) We see from the graph that the mean direction value is nearly independent of the noise level, but the variation of the direction from

the mean increases with increasing noise, as one expects. Let us now increase the tangent window size to 19×19, so that the window contains (roughly) 4 times as many pixels as before (Table 7.1 (b)). Offhand, one expects the standard deviation of the angle calculation to drop by 1/2. Comparing the results of part (a) with part (b) from Table 7.1, one sees that this is not the case. Refer back to Fig. 7.6. In this tangent window, directional information can only be obtained along the edge, which is confined to a narrow strip passing through the center of the window. Any directional information obtained in the larger flat regions on either side of the edge can only be due to noise. Increasing the dimensions of this window to 19×19 will quadruple the flat area, but the area of the strip containing the edge only doubles. So in the larger window we have twice as much true directional information and four times as much noise. The two effectively cancel, as we see from the results in Table 7.1, although for small noise levels the larger window is somewhat preferable.

This effect disappears if directional information is present throughout the tangent window. Table 7.2 gathers the results for the same calculations as Table 7.1, but with a sinusoid replacing the edge. (The results for the larger window are shown graphically in Fig. 7.5.) Specifically, we used

$$h(x, y) = 128 + 48 \sin\left(\frac{\pi}{5}(y \cos\theta - x \sin\theta)\right), \tag{7.5}$$

where the coordinate origin corresponds to the window center and $\theta = 40°$. (This sinusoid has a period of 10 pixels, which roughly approximates the ridge period in

the fingerprint image used in Section 7.5.) With the sinusoid, we see that increasing the window size from 9×9 to 19×19 cuts the standard deviation of the angle calculation by at least one half (and by as much as two thirds for low noise levels).

Let us next consider the tangent consistency score E_N. To be a good predictor of angle reliability, the value of E_N should correlate closely to the standard deviation of the angle calculation. If we compare values of E_N in Table 7.1 (a) against those in (b), we see that for any fixed angle standard deviation level the corresponding mean E_N values in (a) are smaller than those in (b). This is expected and simply means that the larger window can tolerate larger errors and still keep the same angle calculation variance. It is more meaningful to compare the results from the smoothed edge with the sinusoidal at the same window size (i.e., compare Table 7.1 (a) with Table 7.2 (a), and Table 7.1 (b) with Table 7.2 (b)). Doing this we see that for any given angle standard deviation value, the corresponding mean E_N values are within 15% across the tables. For example, suppose we want to consider the calculated tangent angle reliable if the standard deviation is less than 5°. Then for the 9×9 window, the angle is reliable if the value of E_N is less than about 0.47, regardless of whether we look at the single smoothed edge (Table 7.1 (a)) or the sinusoid (Table 7.2 (a)). Thus we can use the value of E_N as a measure of angle reliability, independent of the image structure. Similarly, for the 19×19 window, the 5° standard deviation value corresponds to a E_N score of about 0.68.

Of course, for any one trial the value E_N is a random variable, so its variance is important as well. For example, for the 19×19 window, the standard deviation of E_N with mean value 0.68 is 0.04. Thus if the observed value for E_N is less than 0.6, then with better than 97% probability the noise is in the acceptable range. For smaller windows the variance of E_N is larger, so the estimate is less tight. For E_N with mean value of 0.47 in the 9×9 window case, the standard deviation is 0.07, so to reach the 97% assurance level we need $E_N < 0.34$.

Consider now the contrast score C_N. In the smoothed edge case, comparing the results from the small window to the large window shows that for any given noise level the C_N score drops. This is correct since the larger window contains a larger percentage of flat (zero contrast) area. (Recall that the C_N score is normalized by the window size.) For the sinusoid we see no difference between the C_N scores for the small versus large window, also a correct result. It is somewhat surprising that the C_N score for the small window, smoothed edge (Table 7.1 (a)) is larger than the C_N score for the sinusoids. However, the edge is much steeper than the sinusoid slope, and for the small window the edge fills a significant amount of the total area. Of more importance is the relative insensitivity of the C_N score with respect to the noise level, which is evident from the graphs of C_N provided in Figs. 7.4 and 7.5. Also the standard deviation of C_N is less than 10%, so the C_N score is a reasonable predictor of the underlying (noise-free) image contrast.

As a final comment, note that an edge is not required for the tangent calculation

to return meaningful information. Indeed, the surface may be any for which the level curves are defined (i.e., any inclined surface).

7.4 Curvature vector calculation

7.4.1 Problem formulation

Let us now consider the problem of calculating curvature from level curves. Most previous work on image curvature has been restricted to calculating curvature magnitude from a given boundary curve [39, 57]. In this approach one first uses an edge detection algorithm to locate the image boundary, then pieces the boundary together (using, for example, a chain code algorithm), and then approximates the second derivative of the resulting (1-dimensional) curve. Conversely, our approach uses the results of the tangent calculation only, without any intermediate processing. Moreover, although our procedure works along single edges (especially if one incorporates the extensions suggested in Section 7.4.6), it works best on striated images such as fingerprints, where it can use in the calculations data from several adjacent level curves.

Refer back to Fig. 7.1, and consider the curvature at the point P. We have at our disposal not only the level curve through P, but also the adjacent level curves, which we want to include in our calculation. Notice that the magnitude of the curvature changes as one moves between level curves, increasing as one moves closer to the curvature center, and decreasing as one moves farther away.

Similarly, as one moves along a level curve, the magnitude of the curvature is (or is nearly) constant, but the direction of the curvature vector (towards the center of the curvature) varies. Thus we see that the curvature vector is not constant in any neighborhood of P. The curvature center, however, is (or is nearly) constant. So instead of trying to find an average value for the curvature vector directly, we calculate first an average value for the curvature center. Then the curvature vector can be calculated from the relation $\vec{r}\,''(0) = \overrightarrow{PP_c}/\|\overrightarrow{PP_c}\|^2$, where P_c denotes the curvature center.

The curvature center lies on the line **L** through P that is perpendicular to the level curve tangent vector at P. Since we can calculate the tangent vector using the technique discussed in Section 7.3, we need only calculate the position of P_c on the line **L**. To this end choose another point near P that is not on **L**. If this new point is close enough to P then we expect the two points to share the same curvature center. Construct a line through the second point perpendicular to its tangent. The curvature center P_c is on both lines and is therefore given by the intersection of the two lines. (If the lines are parallel then the curvature is zero and we consider the point P_c to be at infinity.)

Let us expand this idea to a larger region. In Fig. 7.7 there are nine points (P_1, P_2, ..., P_9) with associated tangent vectors ($\vec{u}^1$, $\vec{u}^2$, ..., $\vec{u}^9$) inside a dashed rectangle called the **curvature window**. (The size of the curvature window depends upon the application, but is generally several times larger than the tangent

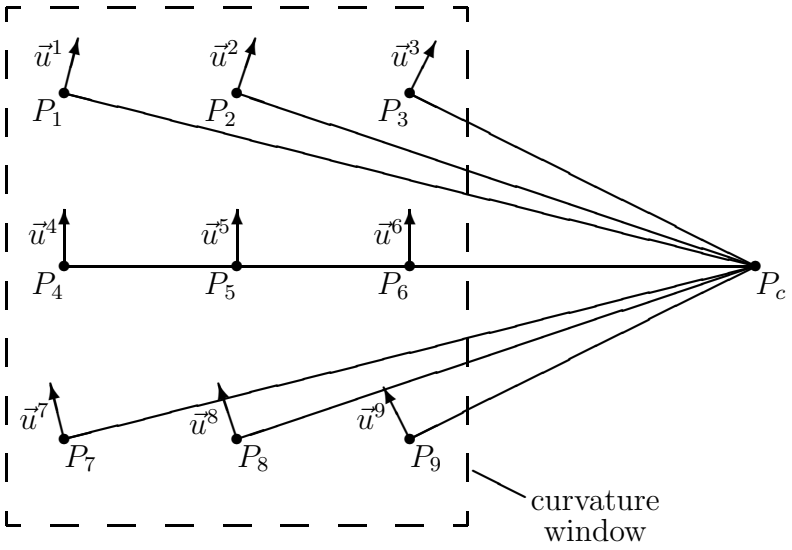


Figure 7.7: Illustration of placement of curvature center point P_c.

window.) Now we want to find a point P_c that can be regarded as the curvature center for all points P_k inside the curvature window. In particular, each radius vector $\overrightarrow{P_kP_c}$ should be perpendicular to the associated tangent vector $\vec{u}^k$, i.e.,

$$\left\langle \overrightarrow{P_kP_c}, \vec{u}^k \right\rangle = 0 \quad \text{for } k = 1, 2, \ldots, 9. \tag{7.6}$$

Then the curvature vector at the center of the window would be $\overrightarrow{P_5P_c}/\|\overrightarrow{P_5P_c}\|^2$.

Of course, it is unlikely that the nine lines will intersect in a point, so we cannot expect there to be a point P_c satisfying the orthogonality conditions of Eq. 7.6. Let us find instead a point P_c such that each radius vector $\overrightarrow{P_kP_c}$ in the curvature window is nearly orthogonal to the corresponding tangent vector $\vec{u}^k$. But how to quantify the concept of "nearly orthogonal"? We consider two different formulations. The first formulation is simple to implement, but is unstable if the tangent vectors are

nearly parallel. The second formulation does not suffer from this instability, but is more difficult to implement.

It is interesting to note that similar problems arise in the application of computer vision to mobile robots [64, 65, 66, 67]. Objects in successive image frames appear to move outward as the robot moves towards them (pure translation). The point from which the motion seems to originate is called the **focus of expansion** (FOE). Lines drawn through the objects in their direction of apparent motion should all intersect at the FOE. This does not occur, of course, due to noise and quantization errors. So given the lines of motion, the problem is to find a point which is as close as possible to all the lines. This is equivalent to the problem that results from using the first orthogonality formulation described below.

7.4.2 First formulation of orthogonality condition

Each pair $(P_k, \vec{u}^k)$ defines a line on which we would like the center to lie—the line $\mathbf{L}^k$ that runs through P_k and is perpendicular to $\vec{u}^k$. Let P_c be a candidate point for the curvature center, and let $d(P_c, \mathbf{L}^k)$ be the perpendicular distance from P_c to the line $\mathbf{L}^k$, as illustrated in Fig. 7.8. The point P_c that we select as the actual curvature center is the point that minimizes the sum of the square of these distances, i.e.,

$$\min_{P_c} \sum_{k=1}^{n} \langle \overrightarrow{P_k P_c}, \vec{u}^k \rangle^2 \tag{7.7}$$

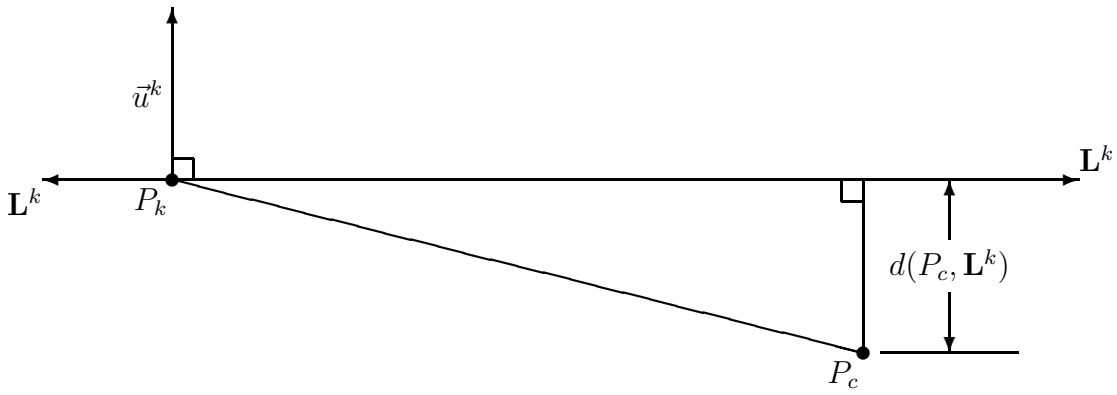


Figure 7.8: Illustration of the first orthogonality formulation for curvature calculation.

where n is the number of points in the curvature window. Here we have assumed that the $\vec{u}^k$ are unit vectors, although we will later relax this restriction. In this formulation (Eq. 7.7), points P_k that are far from P_c have a greater weight in the sum than those close to P_c. The advantage in this is that the tangent vector calculation is more reliable in regions of shallow curvature, so it is natural to weight points away from the curvature center more heavily. Unfortunately, moving the point P_c in towards the curvature window will generally reduce the sum as a whole (since this reduces the size of the vectors $\overrightarrow{P_k P_c}$). Therefore this formulation tends to pull the curvature center inward, increasing the calculated curvature. This is related to the instability problem discussed below.

But let us first derive an explicit solution to Eq. 7.7. Impose a coordinate system on the plane and consider points in the plane as position vectors. Let

$\vec{u}^k = (a_k, b_k)^T$, $\vec{P}_c = (x, y)^T$, and $r_k = \langle \vec{P}_k, \vec{u}^k \rangle$. Furthermore, let $G(\vec{P}_c) = G(x, y)$ be the function being minimized in Eq. 7.7. Then

$$\begin{aligned} G(x,y) &= \sum_{k=1}^{n} (r_k - a_k x - b_k y)^2 \\ &= Ax^2 + By^2 + 2Cxy - 2Dx - 2Ey + M, \end{aligned} \tag{7.8}$$

where $A = \sum_{k=1}^{n} a_k^2$, $B = \sum_{k=1}^{n} b_k^2$, $C = \sum_{k=1}^{n} a_k b_k$, $D = \sum_{k=1}^{n} a_k r_k$, $E = \sum_{k=1}^{n} b_k r_k$, and $M = \sum_{k=1}^{n} r_k^2$.

The minimization of $G(x, y)$ requires the partial derivatives $G_x = G_y = 0$, so

$$\begin{aligned} G_x(x,y) &= 2Ax + 2Cy - 2D = 0 \\ G_y(x,y) &= 2By + 2Cx - 2E = 0. \end{aligned} \tag{7.9}$$

In matrix form this becomes

$$\begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} D \\ E \end{pmatrix}. \tag{7.10}$$

Solving explicitly gives

$$\vec{P}_c = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{AB - C^2} \begin{pmatrix} B & -C \\ -C & A \end{pmatrix} \begin{pmatrix} D \\ E \end{pmatrix}, \tag{7.11}$$

provided that the discriminant $AB - C^2 \neq 0$, in which case this is the unique solution to the minimization problem. On the other hand, the discriminant is zero only if all the tangent vectors $\vec{u}^k$ are parallel, in which case the curvature is zero and we would like the center point $\vec{P}_c$ to be placed at infinity.

Unfortunately, this method is unstable if the tangent vectors are nearly parallel. To see this, consider first the case where the tangent vectors are all exactly parallel.

Then Eq. 7.7 attains its minimum value along the entire line through the center of the curvature window that is perpendicular to the tangent vectors' direction. We would like to select the minimization point to be infinity, but from Eq. 7.7 any point along this line works as well. Now rotate any of the tangent vectors by a small amount, so that the tangent vectors are not all parallel. Then the discriminant $AB - C^2 \neq 0$, so there is a unique solution, but the discriminant is small so the solution is unstable. In particular, even though the solution will lie close to the aforementioned line, it will not necessarily lie near infinity. Moreover, since the formulation penalizes large distances, it is likely that the solution will lie near the center of the curvature window. This would imply a very large curvature, even though the tangent vectors are only slightly perturbed from the parallel condition indicative of zero curvature. Experimental results confirm that this effect is a serious weakness of this formulation.

7.4.3 Second formulation of orthogonality condition

So we need a different formulation for the orthogonality condition. Let $\vec{v}^k$ be a unit vector perpendicular to $\overrightarrow{P_k P_c}$, and define $\vec{e}^k = \vec{u}^k - \langle \vec{u}^k, \vec{v}^k \rangle \vec{v}^k$ as illustrated in Fig. 7.9. Now pick P_c to minimize

$$\min_{P_c} \sum_{k=1}^{n} \|\vec{e}^k\|^2 = \min_{P_c} \sum_{k=1}^{n} \left\langle \vec{u}^k, \frac{\overrightarrow{P_k P_c}}{\|\overrightarrow{P_k P_c}\|} \right\rangle^2 . \tag{7.12}$$

This formulation weights each point equally, but is more complicated than the formulation of Eq. 7.7. In particular, notice that this new formulation suffers from

Figure 7.9: Illustration of the second orthogonality formulation for curvature calculation.

discontinuities at $P_c = P_k$ for each $k = 1, 2, \ldots, n$. However, let us consider the situation where the center point P_c is far from the curvature window, in which case the terms $\|\overrightarrow{P_k P_c}\|$ are nearly identical. Let us place our coordinate system origin at the center of the curvature window and approximate each term $\|\overrightarrow{P_k P_c}\|$ by $\|\vec{P}_c\|$. We then have the simplified formulation

$$\min_{P_c} \sum_{k=1}^{n} \frac{\left\langle \vec{u}^k, \overrightarrow{P_k P_c} \right\rangle^2}{\|\vec{P}_c\|^2}. \tag{7.13}$$

Introducing A, B, C, D, E, and M as in Eq. 7.8 yields the minimization function

$$\tilde{G}(x, y) = \frac{Ax^2 + By^2 + 2Cxy - 2Dx - 2Ey + M}{x^2 + y^2}. \tag{7.14}$$

We want to find the global minimum of this function on the xy-plane. A necessary condition for a local minimum is that the directional derivatives in the radial and angular directions be equal to zero. So let us write $\tilde{G}$ in polar coordinates ($x = r\cos\theta$, $y = r\sin\theta$):

$$\tilde{G}(r, \theta) \;=\; A(\cos\theta)^2 + B(\sin\theta)^2 + 2C\cos\theta\sin\theta$$

$$+ \frac{-2Dr\cos\theta - 2Er\sin\theta + M}{r^2}. \tag{7.15}$$

The partial derivatives are given by

$$\tilde{G}_r(r,\theta) = \frac{2Dr^2\cos\theta + 2Er^2\sin\theta - 2Mr}{r^4} \tag{7.16}$$

and

$$\begin{aligned} \tilde{G}_\theta(r,\theta) &= -2A\cos\theta\sin\theta + 2B\cos\theta\sin\theta + 2C\left((\cos\theta)^2 - (\sin\theta)^2\right) \\ &\quad + \frac{2D\sin\theta - 2E\cos\theta}{r}. \end{aligned} \tag{7.17}$$

Setting $\tilde{G}_r = 0$ and $\tilde{G}_\theta = 0$, simplifying, and converting back to rectangular coordinates produces the requirements

$$Dx + Ey = M, \tag{7.18}$$

$$(B-A)xy + C(x^2 - y^2) + Dy - Ex = 0. \tag{7.19}$$

If M is not zero then at least one of D or E will be non-zero, so these two equations can be combined to yield a quadratic equation in one variable. The (at most) two roots of this quadratic combine with Eq. 7.18 to produce (at most) two candidate points for local minima of $\tilde{G}$. It is also possible that the global minimum of $\tilde{G}$ is attained at infinity. Note that

$$\lim_{r\leftarrow\infty} \tilde{G}(r,\theta) = A(\cos\theta)^2 + B(\sin\theta)^2 + 2C\cos\theta\sin\theta, \tag{7.20}$$

which can be written

$$\lim_{r\leftarrow\infty} \tilde{G}(r,\theta) = (\cos\theta, \sin\theta) \begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}. \tag{7.21}$$

Comparing to Eq. 7.3 and Eq. 7.4 shows that the (boundary) minimum value at infinity is given by the smaller matrix eigenvalue, namely

$$\lambda = \frac{1}{2}\left((A+B) - \sqrt{(A-B)^2 + 4C^2}\right). \tag{7.22}$$

(Notice that Eq. 7.13 implies that $\lambda \geq 0$.) One now compares this value with the value of $\tilde{G}$ evaluated at the (at most) two candidate points found previously. The global minimum is the smallest of these three values, and the curvature center is at the corresponding point.

If M is zero, then the curvature center should be placed at the center of the curvature window (infinite curvature). However, $M = 0$ implies that $D = 0$ and $E = 0$ as well. Introducing this into Eq. 7.14 shows that the minimizing set consists of an entire line through the center of the curvature window. We can therefore expect the same type of instability in this case as we had for the first orthogonality formulation when the curvature was near zero. Moreover, this second formulation will tend to force the calculated curvature center away from the origin because if $M \neq 0$ then $\tilde{G}$ has a pole of order 2 at the origin. Thus the results using the simplified formulation of Eq. 7.13 are unsatisfactory for high curvature situations. (This is expected, of course, since the simplified formulation is based on the assumption that the curvature center P_c is far from the curvature window.)

7.4.4 Combining the two orthogonality formulations

Fig. 7.10 shows the results of applying the two orthogonality formulations to noisy images. For this example we used a curvature window consisting of 9 tangent vector nodes, laid out in a 3×3 grid as in Fig. 7.7. Both the row and column spacing was set to 10 pixels. We fixed a curvature center point P_c and calculated the ideal tangent direction at each of the 9 nodes. The distance of the point P_c from the center of the curvature window determines the magnitude of the curvature vector $\overrightarrow{P_5P_c}/\|\overrightarrow{P_5P_c}\|^2$ (where P_5 is the window center), and the orientation of P_c relative to the fixed 3×3 tangent node grid determines the angular component of the curvature vector.

Both orthogonality formulations produced perfect results on ideal data, so we introduced noise by rotating the 9 tangent directions separately by random amounts. The rotations were determined from a sequence of computer generated zero-mean uncorrelated Gaussian random variables. The noise level was controlled by adjusting the standard deviation of the random variables to 1°, 3°, or 5°. The curvature calculations were performed using the perturbed tangent directions. The random variables were then resampled to generate new tangent perturbations, and the process was repeated. Each point in Fig. 7.10 represents the mean curvature value from 10^5 trials. The abscissa in Fig. 7.10 gauges the original (before noise) curvature, while the ordinate provides the calculated curvature. (The results were found to be independent of the angular component of the curvature vector.) The

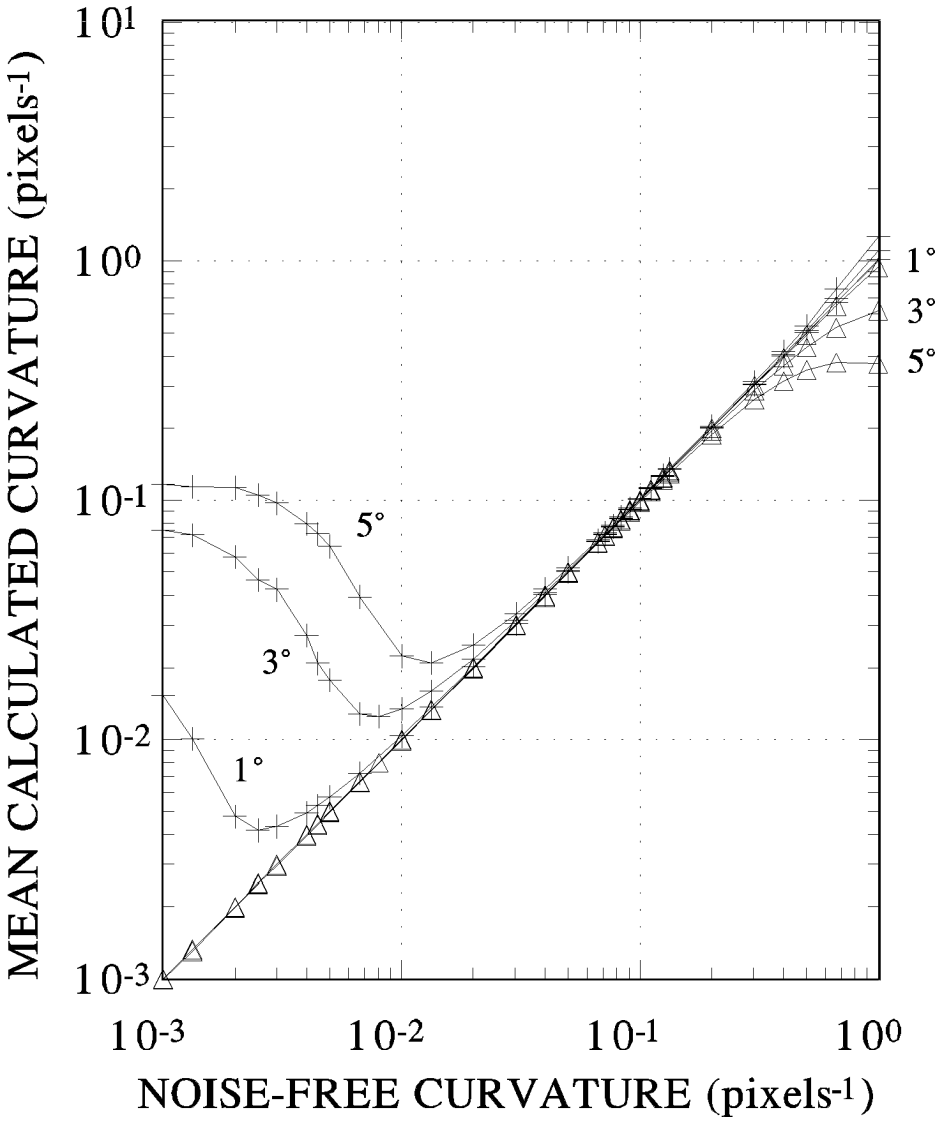


Figure 7.10: Calculated mean curvature as a function of noise-free ("true") curvature and standard deviation (1°, 3°, or 5°) of added Gaussian noise. The results using the first curvature formulation (Section 7.4.2) are marked by pluses, and the results using the second curvature formulation (Section 7.4.3) are marked by triangles. A 3x3 tangent grid with a 10 pixel internal spacing was used for these calculations.

points marked with +'s are the results using the first orthogonality formulation, while $\triangle$'s mark the results using the second orthogonality formulation. There is one curve at each noise level for each formulation.

Ideally the data points in this graph should lie on the line of slope 1 through the origin. However, in the low curvature region (10^{-3} to 10^{-2} pixels^{-1}) the first orthogonality formulation produces exaggerated curvatures from noisy data, as expected. The second formulation performs better across most of the curvature range, although for curvatures near 1 pixel^{-1} the calculated values fall below the ideal values, as also anticipated. Since the first formulation works well for high curvature situations, and the second formulation works well for low curvature situations, it is natural to combine the two formulations. Fig. 7.10 suggests that the first formulation be used for curvatures above about 0.1 pixels^{-1}, and the second formulation below. (The actual curvature values are relative to the size of the curvature window. Here the radius of the curvature window is roughly 10 pixels, so a curvature of 0.1 pixels^{-1} corresponds to the curvature center $\vec{P}_c$ being positioned on the edge of the curvature window.) Therefore, we need an estimator for the curvature that is reliable (with respect to noise) in some range around 0.1 pixels^{-1}. Our experiments suggest that the value of λ from Eq. 7.22 provides such an estimate. This value ranges between 0 if the tangent vectors are parallel to $n/2$ if the tangent vectors are concentric about the center of the curvature window (recall that n is the number of tangent vectors in the curvature window). The

examples presented in Section 7.5 use $\lambda/n = 0.1$ as the method crossover point, i.e., if $\lambda/n > 0.1$ the first curvature formulation was used, otherwise the second.

Fig. 7.11 graphs λ/n under the conditions used to generate Fig. 7.10 (here $n = 9$). We see that $\lambda/n = 0.1$ corresponds to a curvature of about 0.04 pixels^{-1}, a little lower than ideal, but still an acceptable method crossover point. Moreover, if one examines the portion of the graph in Fig. 7.11 near curvature 0.1, one finds a small indentation in the knee of the curve. This is due to discontinuities in Eq. 7.22 as the curvature center $\vec{P}_c$ approaches tangent vector nodes $\vec{P}_k$. Since the tangent vector node spacing is 10 pixels, the curvature center enters the curvature window at curvatures between 0.0707 and 0.1 pixels^{-1}, depending on the angular component of the curvature vector. Unlike the curvature calculations in Fig. 7.10, the values graphed in Fig. 7.11 are dependent on the angular component of the curvature vector, though only in the curvature range between about 0.05 and 0.12 pixels^{-1}. (In Fig. 7.11 the angular component of the curvature vectors is fixed at 57°.) Thus setting the method crossover point at $\lambda/n = 0.1$ is in fact a reasonable choice.

If the minimization error ($G(x, y)$ or $\tilde{G}(x, y)$) is not zero, then the curvature center determined using the first method will lie closer to the curvature window than the curvature center determined using the second method. There will generally be, therefore, some discontinuity in the calculated curvature at the method crossover point. If necessary, one can force continuity by using a weighted average

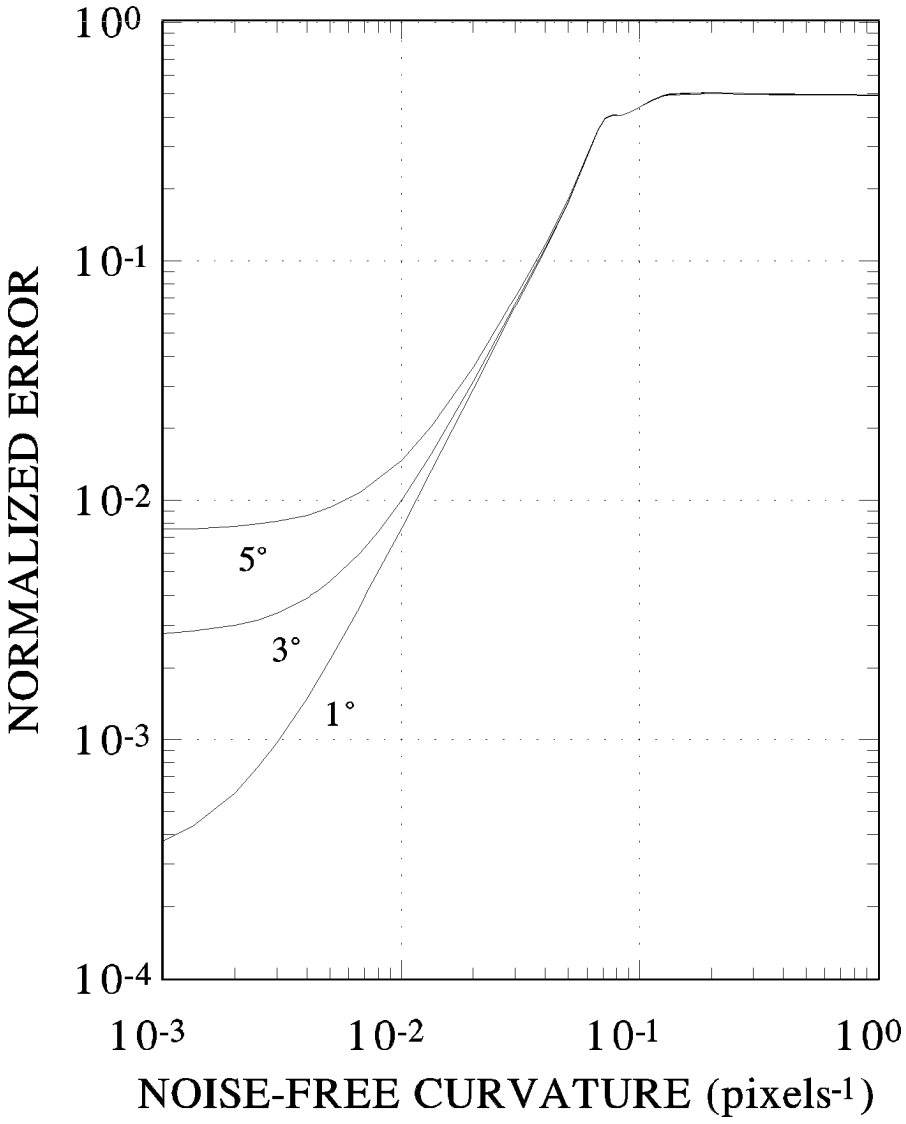


Figure 7.11: Normalized infinity error, λ/n (see Eq. 7.22), as a function of noise-free ("true") curvature and standard deviation (1°, 3°, or 5°) of added Gaussian noise. A 3x3 tangent grid with a 10 pixel internal spacing was used for these calculation. The angle of the noise-free curvature vector with respect to this grid was fixed at 57°.

of the two calculated curvature centers over a transition region. For example, let $\vec{P}_{c1}$ be the curvature center calculated by the first method, $\vec{P}_{c2}$ by the second. Then use

$$\vec{P}_c = \begin{cases} \vec{P}_{c1} & \text{if } \lambda > 0.12n \\ \frac{\lambda - 0.08n}{0.04n}\vec{P}_{c1} + \frac{0.12n - \lambda}{0.04n}\vec{P}_{c2} & \text{if } 0.08n \leq \lambda \leq 0.12n \\ \vec{P}_{c2} & \text{if } \lambda < 0.08n. \end{cases}$$

As an alternative to combining the two curvature formulations, one can explore the possibility of using Eq. 7.12 directly. Minimizing this function requires numerical methods, which may be poorly behaved in high curvature situations (due to discontinuities at $P_c = P_k$). This is left for future study.

7.4.5 Controlled tests

To study the behavior of the combined curvature calculation method in a controlled fashion, we repeated the experiment of Section 7.4.4 using λ/n to automatically select the orthogonality formulation. The results are gathered in Table 7.3. The curvature window consisted of 9 tangent vector nodes, laid out in a 3×3 grid with both row and column spacing set to 10 pixels. As noted before, the dependence of the curvature calculation on the orientation of the curvature vector (directional component) was found to be minimal. Therefore only the dependence on the distance component ("Noise-free Curvature") is listed in Table 7.3.

The second column in Table 7.3 lists the noise level introduced into the data (standard deviations of 1°, 3°, or 5°). For most curvature/noise level combinations, 10^4 trials were performed, but in several instances 10^6 trials were necessary to

Table 7.3: Curvature calculation statistics at various curvature levels and noise magnitudes. Noise was introduced by rotating the tangent directions from a sequence of computer generated zero-mean uncorrelated Gaussian random variables. The curvature units are pixels^{-1}, direction units are degrees.

		Calculation Results		
Noise-free Curvature	Noise Std. Dev.	Curvature Mean Relative Error (%)	Curvature Std. Dev.	Direction Std. Dev.
0.001	1°	0.2	0.0007	0.3
	3°	-1.8	0.0021	1.0
	5°	-1.8	0.0035	1.7
0.005	1°	-0.1	0.0007	0.3
	3°	-0.2	0.0021	1.0
	5°	-1.2	0.0035	1.7
0.01	1°	-0.1	0.0007	0.3
	3°	-0.3	0.0021	1.0
	5°	-1.2	0.0036	1.7
0.03	1°	-0.1	0.0007	0.4
	3°	-0.4	0.0022	1.1
	5°	-1.2	0.0038	1.8
0.04	1°	0.2	0.0008	0.4
	3°	2.0	0.0027	1.2
	5°	5.	0.0050	2.0
0.05	1°	0.2	0.0008	0.4
	3°	1.5	0.0023	1.3
	5°	4.	0.0039	2.2
0.1	1°	0.1	0.0012	0.8
	3°	0.6	0.0036	2.5
	5°	2.	0.0061	4.1
0.5	1°	0.3	0.027	2.5
	3°	2.	0.089	7.8
	5°	8.	0.22	14.
1.0	1°	0.9	0.11	5.0
	3°	11.	0.87	17.
	5°	27.	2.6	36.

provide the desired precision. The resulting statistics are presented in columns 3 through 5.

The results are divided into three columns: curvature mean error, curvature standard deviation, and direction standard deviation. Recall that the result of the curvature calculation is the curvature vector, with magnitude equal to the reciprocal of the curvature radius and direction towards the center of the curvature. The curvature mean error is the difference of the mean value of the calculated curvature magnitude from the curvature before noise, expressed as a percentage with respect to the before noise value. The curvature standard deviation is the standard deviation of the curvature magnitude about its mean value. The difference between the calculated curvature direction mean value and the noise-free direction was insignificant. The standard deviation of the direction component was significant, however, and is listed in the last column of Table 7.3.

The orthogonality formulation used for curvature calculation was selected automatically as detailed in Section 7.4.4, using $\lambda/n = 0.1$ as the method crossover point. In these tests the curvature at the method crossover point was close to 0.04 pixels^{-1}. This means that the method of Section 7.4.2 was used in those cases where the curvature was more than 0.05 pixels^{-1}, and the method of Section 7.4.3 was used where the curvature was less than 0.03 pixels^{-1}. Near 0.04 pixels^{-1}, the algorithm chosen depended upon the noise sample. The trials at noise-free curvature of 0.04 pixels^{-1} had each method selected roughly half the time. The results

in the range below 0.03 pixels^{-1} are most straightforward, so we shall discuss them first.

The calculation errors were nearly independent of curvature in the range between 0.001 and 0.03 pixels^{-1}. The curvature mean value is generally slightly below the true value, and the error increases with increased noise. Notice that in this range the standard deviation of both the curvature magnitude and direction increase linearly with standard deviation of the added noise. No results are listed for curvature less than 0.001 pixels^{-1} because extremely shallow curvature cannot be separated from noise. Even at a curvature of 0.001 pixels^{-1}, the tangent directions in the curvature window vary from the center direction by at most 0.58°. Rotate these directions by noise with standard deviation as small as even 1° and the original curvature is completely lost. The net result is that for extremely shallow curvature (say less than 0.001 pixels^{-1}), the curvature calculation reports a small curvature, but it is not possible to distinguish between two different but very small curvatures in the presence of noise. Along the same lines, the returned direction in the small curvature situation has meaning only modulo 180°. For example, consider Fig. 7.12, which illustrates two curves ((a) and (b)) with small curvature. Curve (a) has curvature direction near 90°, whereas curve (b) has curvature direction near 270°. In the presence of noise, these two curves cannot be distinguished—even though their curvature directions differ by 180°. To allow proper comparisons, it is necessary to restrict the curvature directions to a fixed

180° range, say between 0° and 180°, and to introduce negative curvatures. If the curvature direction is inside the restricted range, no change is made. However, if the curvature direction is outside this range, then we subtract 180° degrees from the curvature direction and multiply the curvature magnitude by -1. For example, with this modification the curvature vector of curve (a) is unchanged, but curve (b) would have curvature direction near 90° and a (small) negative curvature.

As the curvature increases from 0.03 through 0.04 pixels^{-1}, the curvature calculation method used shifts from the method of Section 7.4.3 to the method of Section 7.4.2, until for curvatures ≥ 0.05 pixels^{-1} the method of Section 7.4.2 is used almost exclusively. We see that the variance in the direction result increases as the curvature center point P_c moves in towards the curvature window. The curvature magnitude results, on the other hand, actually improve as the curvature increases, until the curvature reaches 0.1 pixels^{-1}. The dimensions of the curvature window are such that at this point the curvature center is just inside the curvature window. After this point the results degrade. In high curvature situations there are problems analogous to the problems at very low curvature. For example, suppose the true curvature is 1 pixel^{-1}. Then the curvature center is only 1 pixel away from the center of the curvature window. Since the curvature magnitude and direction is measured relative to the window center, small changes in the location of P_c have dramatic changes in the curvature values. For example, a location error of only 1 pixel can move P_c from its original location to the exact center of the

curvature window, at which point the curvature magnitude is infinite and the curvature direction is indeterminate. However, this error cannot change the calculated curvature from a large value to a small value. We can say that the curvature is very large, but we cannot say exactly how large. That is to say, the distribution of the curvature magnitude about the mean is not symmetric. Consider, for example, the test from the last line of Table 7.3, where the curvature mean value is 1.27 pixels^{-1} with standard deviation of 2.6 pixels^{-1}. In this test only 27% of the trials were above the mean value, indicating that the curvature values beneath 1.27 pixels^{-1} were mostly not far from that value. Indeed, the median value for this test was 0.92 pixels^{-1}, and fewer than 4% of the values were below 0.5 pixels^{-1}.

7.4.6 Extensions

The contrast (C_N) and consistency (E_N) measures of the tangent direction from Section 7.3.4 give a measure of the reliability of the directional information. It is natural, therefore, to weight the tangent directions in the minimization problems of Eq. 7.7, 7.12, and 7.13 according to their reliability. In particular, one can introduce weights w_k^2 into Eq. 7.7 by

$$\min_{P_c} \sum_{k=1}^{n} w_k^2 \langle \overrightarrow{P_k P_c}, \vec{u}^k \rangle^2. \tag{7.23}$$

For example, we can set $w_k = 0$ if the contrast score C_N for tangent vector u_k is too small, and otherwise set $w_k = 1 - E_N$.

Let us rewrite Eq. 7.23 as

$$\min_{P_c} \sum_{k=1}^{n} \langle \overrightarrow{P_k P_c}, w_k \vec{u}^k \rangle^2$$

where each weight w_k becomes the magnitude component of the tangent vector $\vec{u}^k$. Although we originally specified in Eq. 7.7 that the vectors $\vec{u}^k$ be unit vectors, this restriction is not used in the derivation of the solution. Working through the derivation with $\vec{u}^k$ replaced by $w_k \vec{u}^k$, we find that the solution to Eq. 7.23 is given by Eq. 7.11 but with $A = \sum_{k=1}^{n} (w_k a_k)^2$, $B = \sum_{k=1}^{n} (w_k b_k)^2$, $C = \sum_{k=1}^{n} w_k^2 a_k b_k$, $D = \sum_{k=1}^{n} w_k^2 a_k r_k$, $E = \sum_{k=1}^{n} w_k^2 b_k r_k$, and $M = \sum_{k=1}^{n} (w_k r_k)^2$. Weighting can be added to Eq. 7.12 and 7.13 in a similar fashion.

One can generally improve the curvature results by averaging the tangent vectors before the curvature calculation takes place. For example, consider images for which the tangent vectors form a smooth flow, with a handful of singular points. One wants to average the tangent vectors in the smooth regions without losing the singular points. Kawagoe and Tojo [52], working on fingerprint classification, used an effective relaxation technique along with singular region detection. Alternately, one may be able to adapt the convex projection technique of Simard and Mailloux [68] to this problem.

7.5 Examples

In our experimental system images are input from a standard video camera and digitized using a Data Translation IBM-AT compatible frame grabber. The digi-

tized images have up to 256 grey levels and a full screen image is 512 pixel columns by 480 pixel rows. The pixel aspect ratio is such that a rectangle 4 columns wide by 5 rows high appears square. Since this makes the sample node grid non-square, a modification is required to the point normal calculation. The details are straightforward, however, and are left to the reader.

7.5.1 Tangent examples

One sample use of the tangent calculation is in fingerprint identification. Ridge flow directions must be established in order to locate both minutiae (ridge endings and bifurcations) and flow singularities (cores and deltas). An inked fingerprint is shown in Fig. 7.13. Notice the hole on the right side and the poor contrast due to over inking at the top and lower left side. We applied the algorithm of Section 7.3 to this image using tangent windows of 19 rows by 15 columns. (When the pixel aspect ratio of the frame grabber board is taken into account, this window is approximately square.) The window size is chosen large enough so that noise can be controlled, but small enough so that curvature inside the window is negligible. We have found this size window, which will generally contain 2 ridges, to work well in practice.

Fig. 7.14 shows the results of the tangent calculation for each point on a 10 row by 8 column grid. (Compare also to examples in [58, 59].) The corresponding tangent direction is indicated by a line segment unless the contrast or consistency

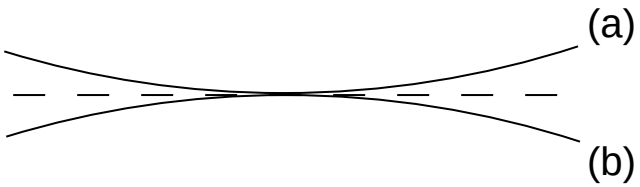


Figure 7.12: Illustration of two shallow curves that are indistinguishable in the presence of noise. The curvature vector direction in such situations has meaning only modulo 180°.

Figure 7.13: Image of an inked fingerprint. Notice the missing hole on the right hand side and the poor contrast at the top.

is poor (refer to Section 7.3.4), in which case the point is left unmarked. This gives a rough indication of which regions of the print are of good quality (marked) and poor quality (unmarked). In our actual experimental system the good regions are further classified by encoding the line segments with color. Notice that the hole on the right side is left unmarked, as are sections of the top and bottom of the print. Also notice that there are no tangent directions marked outside the fingerprint (in the background on the extreme left and right). We see that in addition to giving the ridge flow direction, the tangent calculation can also be used for fingerprint segmentation.

A more general use of the tangent calculation is for adaptive filtering. For example, if one wishes to enhance edges in an image, then knowledge of the tangent direction allows one to select an appropriate orientation specific filter. In the following examples we used 8 convolution-type filters (one each for 0°, 22.5°, 45°, ..., 157.5°). The kernel of the filter for the direction 22.5° was

$$\begin{array}{rrrrrrr} 0 & 0 & -8 & -6 & 0 & 0 & 0 \\ -1 & -9 & -8 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 2 & 11 & 18 & 7 \\ 0 & 2 & 11 & 18 & 11 & 2 & 0 \\ 7 & 18 & 11 & 2 & 0 & 0 & 0 \\ 4 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -8 & -9 & -1 \\ 0 & 0 & 0 & -6 & -8 & 0 & 0 \end{array}$$

with the resulting value scaled via division by 66. Each filter exhibits low-pass (smoothing) behavior in the specified direction and high-pass (sharpening) behav-

Figure 7.14: Tangent directions calculated on a 10 row by 8 column grid using a 19 row by 15 column tangent averaging window. Unmarked regions indicate either the contrast or the consistency is poor.

ior in the perpendicular direction. We then used the following procedure. At each point in the image, calculate the tangent direction along with the contrast and consistency scores. If the contrast and consistency are sufficiently good (as determined by C_N and E_N), then select and apply the filter that has orientation corresponding to the tangent direction. If the contrast or consistency of the region is poor, then leave the image value at that point untouched.

The results of apply this procedure to the fingerprint of Fig 7.13 is shown in Fig. 7.15. The tangent averaging window was 19 rows by 15 columns. Regions were considered to be of poor quality (and hence left untouched) if the normalized contrast score C_N was less than 0.07 or if the consistency error E_N was larger than 0.7. Repetition of the filtering allows good regions to expand into poor regions. (This effect was also noticed by Peli [54], who obtained similar results using frequency domain analysis and filtering.) Fig. 7.16 shows the results after 8 iterations of the filter. To prevent over-filtering, the first 6 passes filtered only those points that were not modified (due to poor contrast or tangent consistency) in the preceding passes. After the sixth pass most noise had been eliminated from the image, so for the final two passes a smaller 9×7 tangent window was used for increased directional precision.

As another example of the use of this directional enhancement, consider Fig. 7.17, which is an image of a porous membrane obtained from a scanning electron microscope. For experimental work we needed to determine the pore volume fraction.

Figure 7.15: Result of one pass of the directional filter. The tangent window was 19 rows by 15 columns. Regions were left unprocessed if the normalized contrast score was less than 0.07 or if the normalized consistency error was larger than 0.7.

Figure 7.16: Result after 8 passes of directional filter. Each of the first 6 iterations only modified pixels untouched by preceding passes. The last two passes modified low contrast regions using a 9 row by 7 column tangent window. The smaller window allows for the capture of tangents in high curvature regions.

Thresholding the image is made difficult by the vagueness of the pore boundaries. Two passes of this directional filtering gives the image in Fig. 7.18. Thresholding this image yields pore fractions that agree to within 1% with values obtained by manual inspection.

7.5.2 Curvature examples

Regions of high curvature denote singular regions in images. Refer back to the fingerprint image of Fig. 7.13. The most notable regions are the core (near the center) and the two deltas (below and on either side of the core). Fig. 7.19 illustrates the results of the curvature calculation. (Compare to [58].) The curvature is displayed on a mesh of 20 rows by 16 columns. (This mesh size was chosen for presentation purposes; a denser grid makes the results more difficult to read.) Curvature is not calculated in poor regions (as defined by the tangent calculations), which explains the lack of markings in the hole on the right side and in the lower left corner. Each curvature window consisted of a 3×3 array of tangent vectors from Fig. 7.14. If the calculated curvature radius is less than 150 pixels (roughly 1/3 the image height), then the calculation window center is marked with a '+' and a radial line is drawn to the curvature center, marked with a '$\circ$'. If the curvature radius is larger than 150 pixels, then a short line segment is drawn at the calculation window center in the direction perpendicular to the calculated curvature direction. (Most of the curvature radii are quite large; if every radial line were drawn then the image would

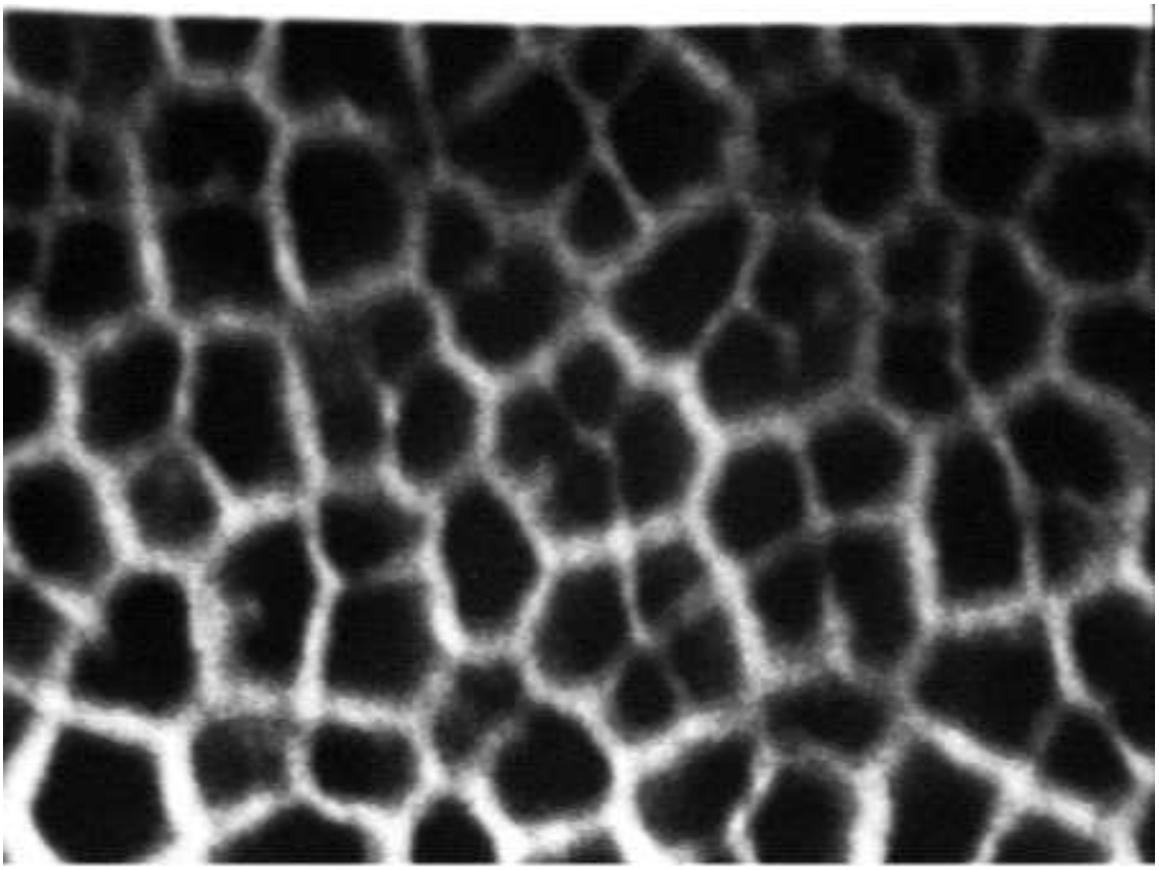


Figure 7.17: Scanning electron microscope image of a membrane. Dark areas are pores in the membrane.

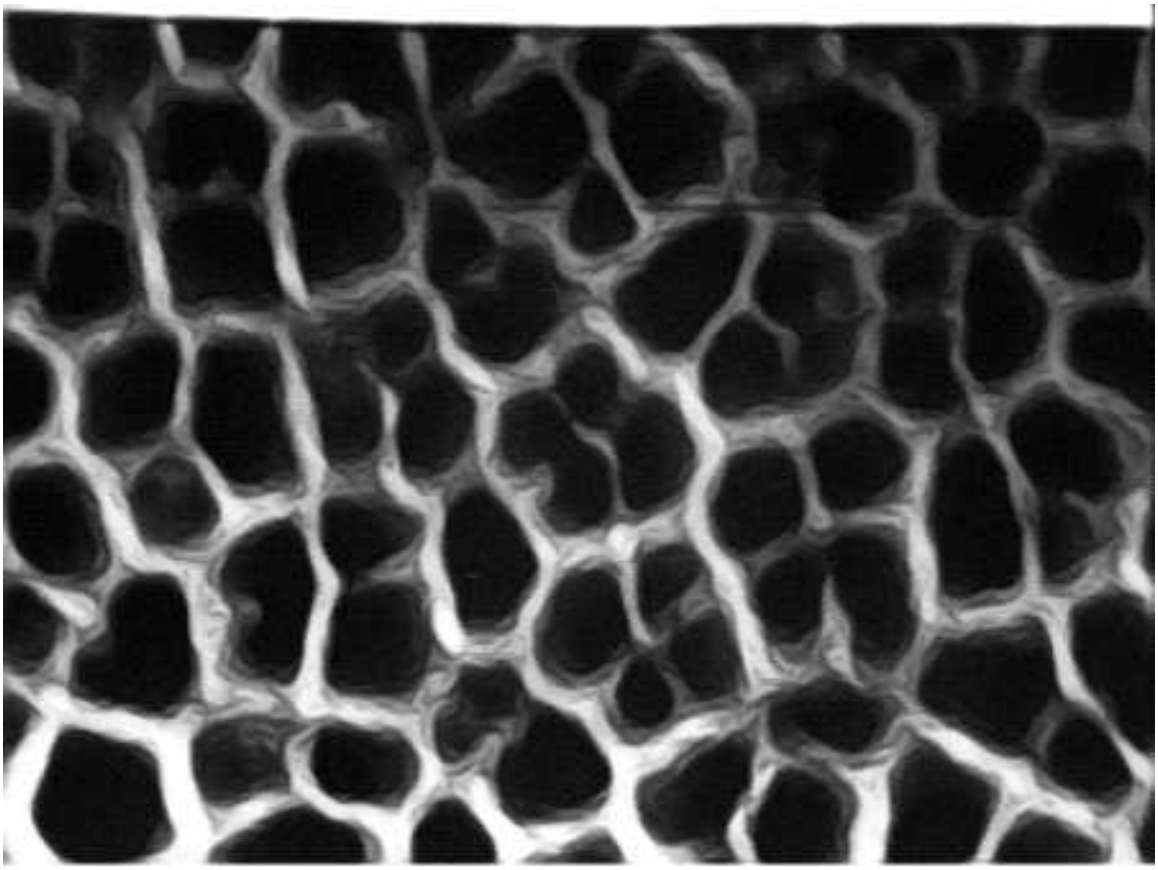


Figure 7.18: Membrane image after two passes of directional filter using 15 row by 11 column tangent window. Thresholding this image gives pore volume fraction in agreement with results from manual inspection.

be a tangle of lines.)

Careful examination of this figure reveals that the regions with the largest curvature (shortest radial lines) are the deltas and the areas just above and below the core. This is borne out in the curvature magnitude contour plot of Fig. 7.20. Actually, there is an implicit assumption in the curvature calculation that the tangent flow is continuous. This assumption is violated across the fingerprint deltas, but nonetheless the algorithm returns the desired result: large curvature.

The curvature can be used to locate features in a wide array of images. Fig. 7.21 shows the curvature results for a section of a printed circuited board. The curvature accurately marks the circular wire pads and the resistor elements. In addition, the tangent calculation has distinguished the circuit trace boundaries from the featureless background board.

As a final example, Fig. 7.22 shows curvature for a low contrast radiograph of a steel part with 4 shallow, oval slots (simulating flaws). Due to the poor contrast, only 3 of the 4 slots are definitely detectable (the second slot from the left is detectable mainly due to the presence of the other three slots). High curvature locates the 3 detectable slots.

7.6 Summary

In this chapter we have presented algorithms for the extraction of tangent directions and curvatures of level curves of images. The effects of noise on these algorithms

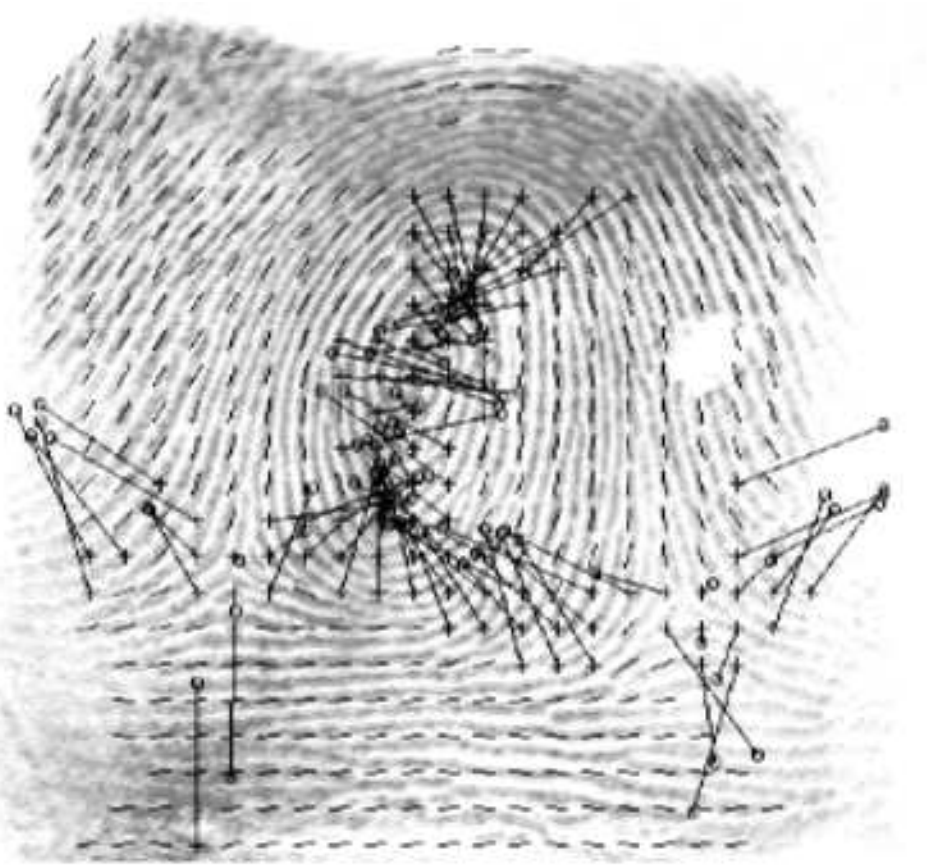


Figure 7.19: Fingerprint overlaid with results from curvature calculation. Points with curvature radius of less than 150 pixels are marked with a '+', and a radial line is drawn from the point to the calculated curvature center, marked with a '∘'.

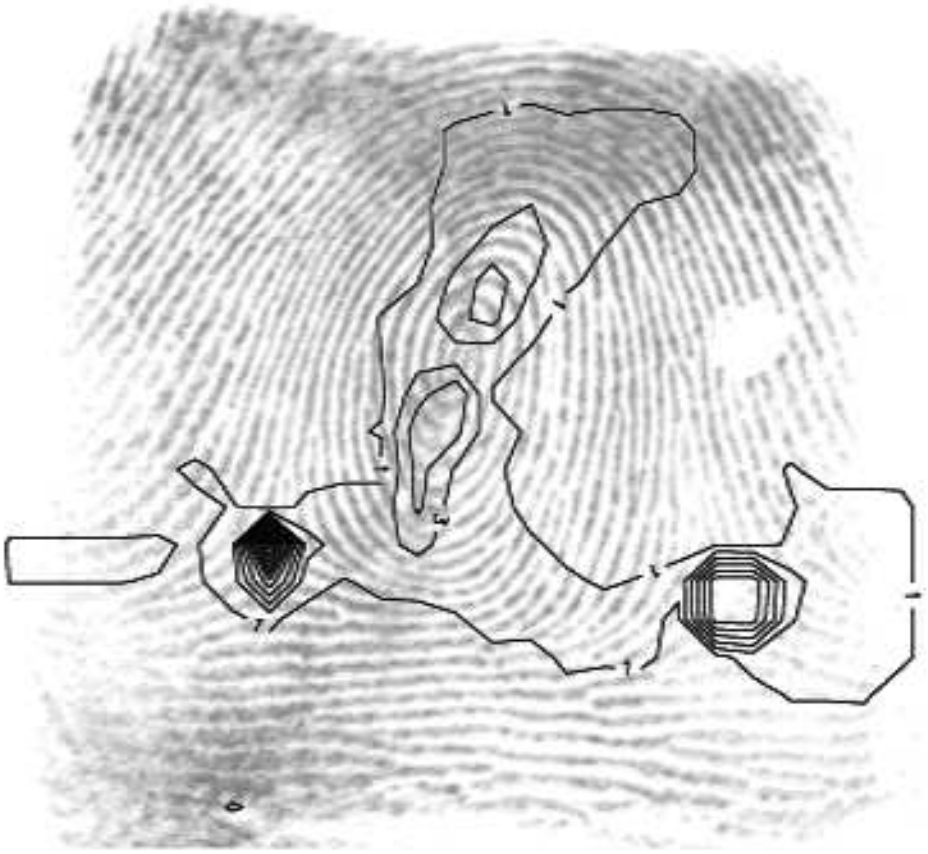


Figure 7.20: Curvature magnitude contours (units: $0.01 \times (\text{pixels})^{-1}$) overlaid on the fingerprint image. The high curvature values correctly mark the fingerprint core in the center of the image and the deltas below on either side.

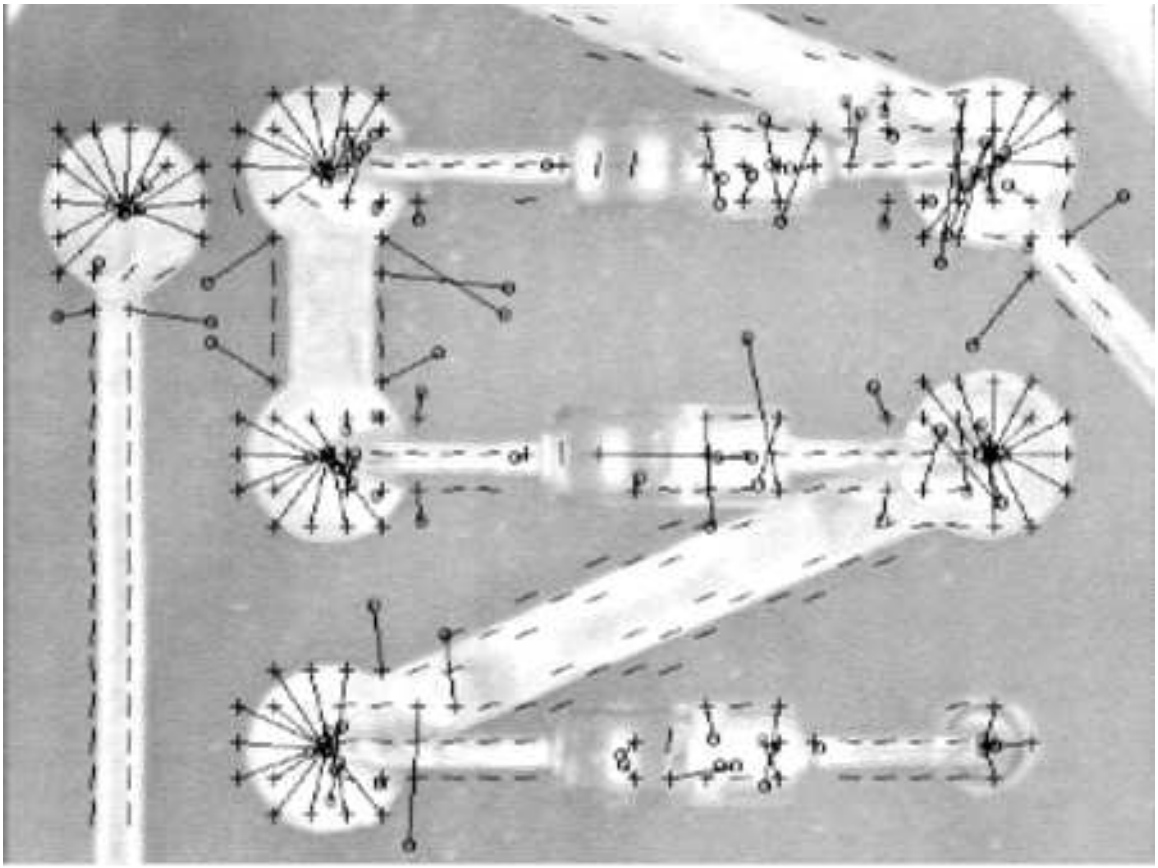
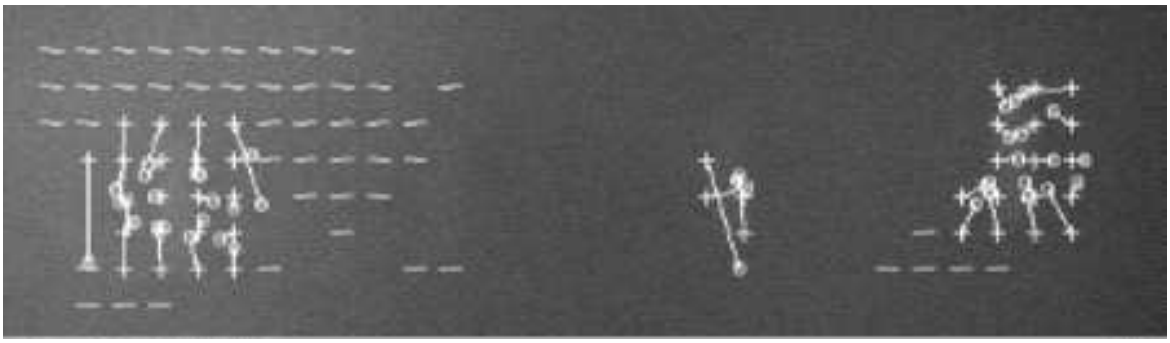


Figure 7.21: Section of a printed circuit board overlaid with calculated curvature. High curvature locates circular pads and resistor elements.

(a)



(b)

Figure 7.22: (a) Radiograph of a steel specimen with 4 oval slots (which simulate flaws). (b) Curvature overlay. High curvature locates 3 of the 4 slots.

were studied under controlled conditions using simulated data and the usefulness of the tangent and curvature information was shown by examples with several real images.

The tangent direction is extracted by a least-squares minimization over the surface normals (calculated for each 2×2 pixel neighborhood) in the averaging window. Even with a signal-to-noise ratio (defined as the ratio of the variance of the original image to the variance of the noise) as low as 10, a single edge passing through the center of a 9×9 window results in a tangent angle calculation having mean value accurate to within $3°$. The standard deviation of this calculation is less than $3.5°$, a value that drops to $1.3°$ when the edge is replaced with a sinusoid passing through a 19×19 window. The minimization error from the tangent calculation can be used to estimate the reliability of the calculated tangent direction. This error grows with the variance of the calculated tangent direction, and has a standard deviation $< 10\%$ for the larger 19×19 window.

The usefulness of these calculations were illustrated by using the tangent information to select orientation sensitive filters for edge enhancement on images of a fingerprint and of the microstructure of a porous membrane. The contrast and consistency scores from the tangent calculation also effectively segmented the fingerprint and a section from a printed circuit board.

Unlike most previous work on this topic, the curvature calculation does not require a (single) parameterized curve, but works instead directly on the tangent

directions across adjacent level curves. The curvature is found by fitting concentric circles to the tangent directions via least-squares minimization. Accuracy and reliability were studied by controlled tests with simulated noise. The curvature information can be used for feature detection and identification, as illustrated by results showing high curvature locating cores and deltas on a fingerprint, circular pads and circuit elements of a printed circuit, and slots simulating flaws in a radiograph of a steel part.

CHAPTER VIII

Feature extraction on printed circuit boards

This chapter and Section 10.1 profiles a feature based method for flaw detection on printed circuit boards. The material is culled from work done by the author with Alan Sprague and published in Computer Vision, Graphics, and Image Processing [1]. In that work an algorithm was developed that performed both a reference comparison based on extracted features and a separate minimal wire width check. The wire width check is beyond the scope of this dissertation, and will not be presented here. The interested reader may refer to [1] for details.

8.1 Method Outline

The reference comparison begins with a digitized image of a printed circuit board. It is assumed that the image has been previously thresholded into light and dark pixels representing circuit traces and substrate respectively. A row by row scan of this binary image is made and the light pixels are grouped together to form a compact circuit representation we call the **segment graph**. The segment graph is

then analyzed to remove extraneous elements, resulting in the **reduced segment graph**. A ruled based procedure uses the reduced segment graph to extract feature points representing wire intersections, ends, and isolated pads. Circuit defects such as wire breaks, bridges (shorts), and pores inside wires also generate feature elements. A one-to-one comparison is made between the features extracted from the board under test to the features extracted from an reference (defect free) circuit board. (The reference board could be a computerized image generated by circuit design software.) Any unmatched features, whether on the test board or the reference, indicate a manufacturing defect on the test board.

8.2 The Segment Graph

The segment graph is the central tool in our extraction of features from the printed circuit board. It is formed during the row by row scan of the board. Since metal traces on the board appear as connected groupings of light pixels, any metal trace present in a row scan displays itself as a connected run of light pixels. If two runs of light pixels from successive rows overlap, then those runs correspond to connected metal traces. We say that two runs $R(j)$ and $R(j+1)$ of light pixels from successive rows j and $j+1$ **match** if $R(j)$ overlaps $R(j+1)$ but overlaps no other run of light pixels in row $j+1$, and also $R(j+1)$ overlaps no run of light pixels in row j other than $R(j)$. A **segment** is defined to be a maximal collection of matching runs of light pixels. Refer to Fig. 8.1 (a) for an illustration of these definitions. In

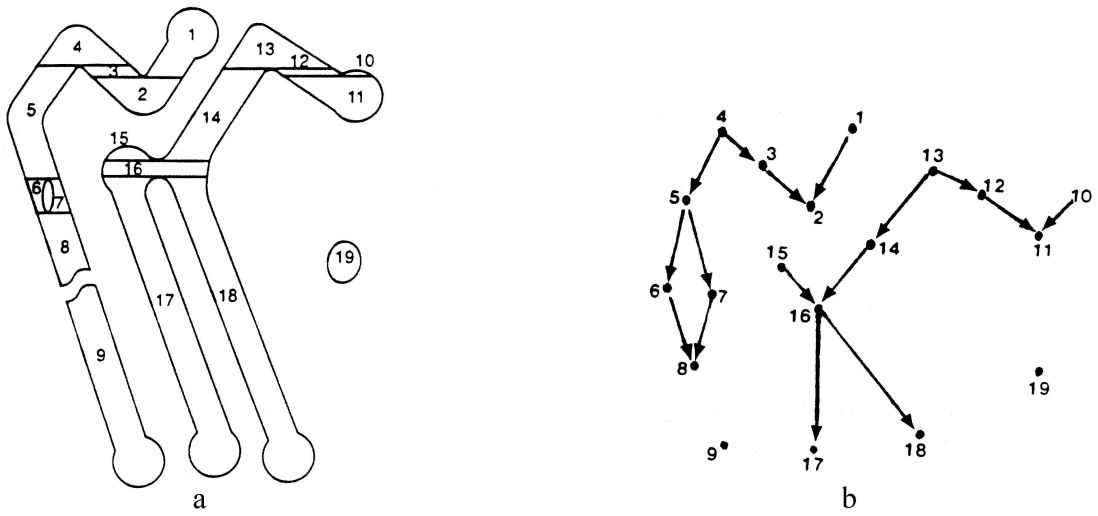


Figure 8.1: (a) Fragment of electric circuit; (b) The corresponding segment graph representation. (From [1])

this figure horizontal lines separate individually numbered segments. Consider for example segment 4 in the upper left side of the figure. It is a continuous section of metal which extends down to the level where it splits into two new segments numbered 5 and 3. The bottom run of light pixels in segment 4 overlaps the top runs of light pixels in *both* segment 5 and segment 3. Therefore the bottom run of light pixels in segment 4 has no *matching* light pixel run in the succeeding row, and thus marks one end of segment 4.

The **segment graph** is a directed graph embodying the relative positions and connections between the segments. Each segment is a node (vertex) in the graph. The edges (arcs) of the graph denote segment adjacency, i.e., electrical conductivity. There is an arc connecting two nodes if the bottom run of light

pixels from one of the represented segments overlaps the top run of light pixels from the other. The arc is directed from the upper segment to the lower one. (The graph terminology is standard; refer to [69].) For example, refer to node 4 in Fig. 8.1 (b). This node corresponds to segment 4 in (a). Recall that the bottom pixel run from segment 4 overlaps the top pixel runs from segments 5 and 3. This connectivity is incorporated into the segment graph as directed arcs from node 4 to each node 5 and node 3.

8.3 Feature extraction

Included in the data structure realizing the segment graph is the physical locations (row and column) of each segment top and bottom. This information is needed not only for feature matching, but also for feature extraction. The first step in feature extraction is the removal of spurious segments, which we call **coastal**. A segment is coastal if it is connected to exactly one other segment and its height (the distance from the top pixel run to the bottom pixel run) is less than a predetermined value called the **coastal height threshold**. (Particular examples of coastal segments are irregularities along the top and bottom of horizontal wires.) In our work the coastal height threshold was set to 3 times the minimal allowable wire width. (The minimal allowable wire width is used by the wire width checking algorithm in the detection of too narrow wires. The wire width checking algorithm is not discussed here; refer to [1].) In Fig. 8.1 only segments 10 and 15 are coastal.

Coastal segments are repeatedly removed and the segment graph updated to form the **reduced segment graph**.

Features are extracted from the reduced segment graph by a rule based procedure. There are 3 main feature types: ends, bifurcations, and dots. These are illustrated in Fig. 8.2. The rules for extraction of these features depend upon segment connectivity and height. (For purposes of feature extraction a segment is considered short if the distance from the top to bottom row is less than a predetermined value. In our work this value was set to 3 times the minimum wire width.) Feature extraction for short segments depends upon the total number of connections to that segment. A short segment with no connections receives a "dot" designation. See for example segment 19 of Fig. 8.1. If a segment is short and has three connections, it generates a "bifurcation", as segment 16 in Fig. 8.1. (Note that segment 15 is a coastal feature and so is not present in the reduced segment graph.) On the other hand, feature extraction for tall (i.e., not short) segments depends on the number connections at each end (top or bottom). If one end of a tall feature has no connections, then an "end" type feature is placed at that end. See for example the bottom of segment 18. The complete set of rules for feature extraction is given in Fig. 8.3.

Defects on the printed circuit board will introduce, modify, or delete features as compared to the test board. Fig. 8.4 illustrates some possibilities. A break in a wire trace introduces two ends (lower right corner), while an overetched circuit pad

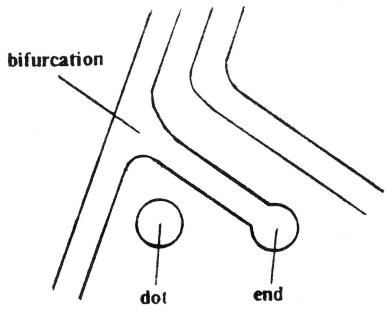


Figure 8.2: Printed circuit board feature definitions. (From [1].)

number of connections	feature type and number
0	1 dot
1	1 end
k ($k \geq 2$)	$k-2$ bifurcations

(a)

number of connections at this end	feature type and number at this end
0	1 end
k ($k \geq 1$)	$k-1$ bifurcations

(b)

Figure 8.3: Feature extraction rules for short segments (a) and tall segments (b). (After [1].)

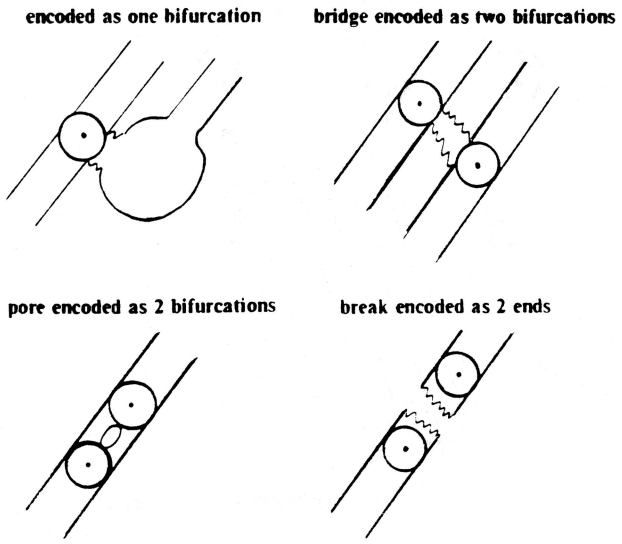


Figure 8.4: Feature point encoding of printed circuit board defects. (From [1].)

can change an end type feature to a bifurcation (upper left corner). The reader may entertain other examples as well. Fig. 8.5 shows the features extracted from the circuit of Fig. 8.1, including two false ends caused by a break and two false bifurcations caused by a pore in the leftmost wire trace.

8.4 Comments

The features extracted using the method described above can be matched against a reference set for the determination of defects. An algorithm for such a comparison is detailed in Section 10.1, including sample results using features provided by the extraction method of this chapter. It is, however, worthwhile to note some

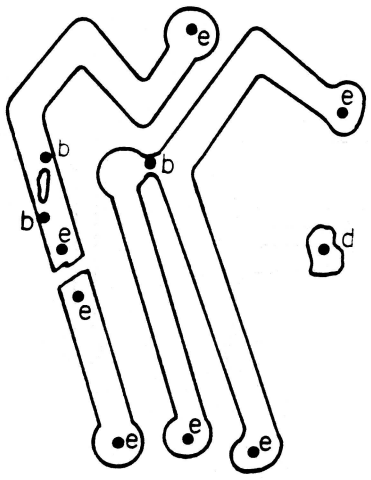


Figure 8.5: Feature points extracted from circuit of Fig. 8.1. (From [1].)

shortcomings of the feature extraction method at this time.

Please refer first to the metalization patterns of Fig. 8.6. Here we see two electrically different patterns which are both encoded as four end type features. Since the extracted feature set is the same, the matching algorithm cannot detect any difference between them, and the fault will go undetected. To detect such (presumably unlikely) occurrences, the algorithm would need to be modified to include component information along with the feature type. (Component here refers to electrically or graph theoretically connected material.)

There is also a difficulty with the extraction of features at X-shaped intersections. The above algorithm extracts such an occurrence as two bifurcations. However the matching algorithm requires some minimal spacing between feature points on the reference board. To handle this difficulty it is necessary for the

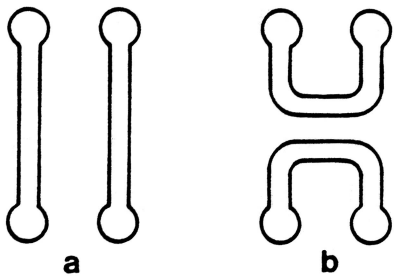


Figure 8.6: Two metal trace patterns that are electrically distinct but which generate the same feature points. (From [1].)

extraction program to group together such bifurcation pairs as a single feature element.

A final point to note is that the matching algorithm requires special "alignment" features to place the test board in proper position with respect to the reference board. These alignment points presumably have a topology not found elsewhere on the board. The relaxation of this requirement is discussed in Section 10.1.

CHAPTER IX

Feature extraction from local image topology in fingerprints

9.1 Overview

In this section we give an overview of the processing that we perform. We then note some consequences of the method outlined, and some complications. Following that is a detailed account of the minutiae extraction process with references to subroutines.

A fingerprint is digitized as an image of 480 rows and 512 columns. Each pixel has one of 256 intensity levels. The objective of processing is to extract minutiae on the fingerprint. To do this we do not process the entire fingerprint at once; rather, we process one much smaller region, called a 'subregion', at a time. This is done because it is far easier to extract minutiae on a region in which all ridges (and all valleys) flow left and right, than on a region where the direction of flow is highly variable.

To extract minutiae on a subregion we perform the following processing steps.

1. Compute the average flow direction in the subregion. Flow is defined as the direction of the ridge tangent vector. The method for extraction of this information is detailed in Chapter VII.

2. Introduce a superwindow containing the subregion. The superwindow is a rectangle, whose boundaries are parallel and perpendicular to the direction of flow. The subregion is to be entirely within the superwindow.

3. Rotate the superwindow so that flow within it is horizontal (and its boundaries are horizontal and vertical).

4. Extract minutiae on the superwindow.

5. Extract local topology information (average ridge width and curvature).

Before giving a more detailed exposition, we note the following consequences and complications on the foregoing description.

1. Since decisions regarding minutiae near the edge of a superwindow are unreliable, some decisions near the edge of a subregion are likely to be unreliable. Therefore, subregions are designed to overlap so that minutiae in the overlap region (i.e., minutiae near the subregion edge) should be extracted in multiple subregions.

2. Flow is computed for the entire image before any other processing is started. This cuts down on redundant computation, since superwindows overlap greatly.

3. It may be that the direction of flow in one part of a subregion is much different than the direction of flow in another part of it. To compensate for this the program will automatically make the subregion (and the superwindow) smaller in the troublesome area, or process the subregion several times, with the flow direction declared to be different each time; the idea is that in each part of the subregion the declared flow will be roughly correct one of the times.

9.2 The feature extraction window and superwindow

The image is divided into a grid of overlapping **subregions**. The grid of subregions is pictured in Fig. 9.1. In regions where the flow information is highly variable (e.g. singular points such as the core and delta) the subregion grid is overlaid by a finer grid approximately 1/4 the size, i.e., a subregion containing highly variable flow is replaced by four smaller regions. These smaller regions are called **reduced subregions**. Ideally, in each of these reduced subregions the flow variability will be small. In reality this is often not the case. When the variability is still high the reduced subregion is processed several times, with the flow direction declared different each time.

To process a subregion, a slightly larger region, called a superwindow, is utilized. For a given subregion (from the subregion grid) the corresponding superwindow is defined by two conditions: (1) the edges of the superwindow are parallel and

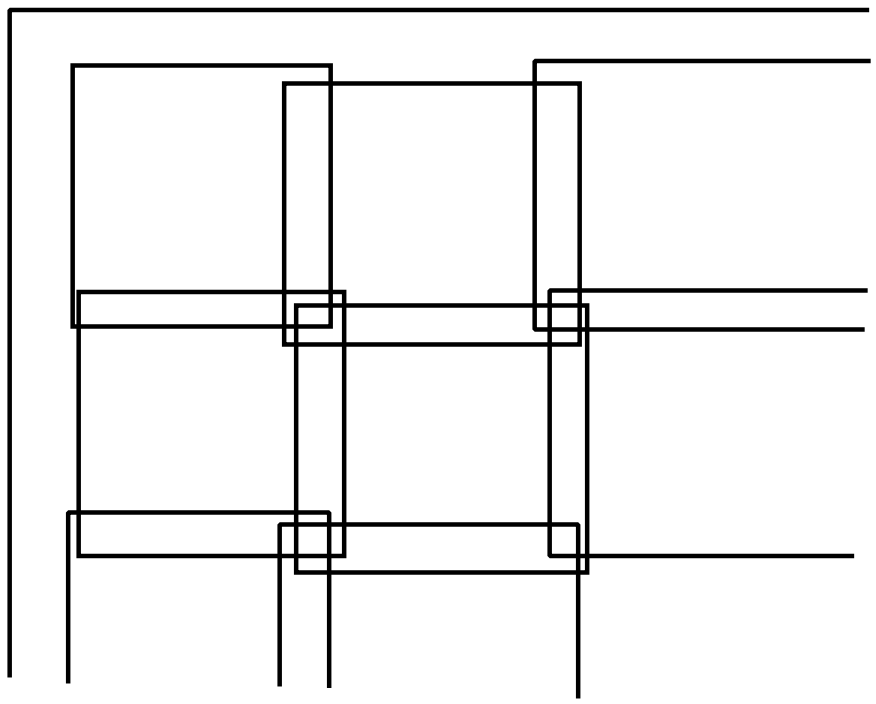


Figure 9.1: Schematic illustration of the subregion grid. The edges of the subregions are offset to aid visibility.

perpendicular to the flow in the subregion, and (2) a minimum border distance from any point in the subregion to an edge of the superwindow is maintained. A border is required so that at any point in the subregion, where for example we are examining a possible minutia, we will process a neighborhood completely surrounding the point.

The data in the superwindow region is extracted from the digitized image, rotated, and stored in a temporary processing array. Of course the pixel values in the original image are defined only on an integer grid. After rotation the pixel locations do not in general lie on the same grid. The rotation routine simply shifts the rotated pixel values to the nearest integer grid location. This allows the rotation to be swift, but the resulting rotated image has a ragged appearance. The rotated image is filtered to remove this artifact.

9.3 Flow specific parametric filtering

The rotated image is filtered both for the removal of rotation artifacts and also for general image enhancement. Three filters are available: a lowpass filter, a weak filter, and a strong filter.

lowpass filter The lowpass filter is a convolution filter with the following kernel:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This is an omni-directional smoothing filter. This is used when no directional emphasis is desired.

weak filter The weak filter is a 5x1 convolution filter to accomplish horizontal smoothing. The filter has kernel:

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

Note that this filter is applied to a superwindow after it is rotated, i.e., the dominant flow direction is horizontal. Thus this filter smooths along each ridge and along each valley.

strong filter The strong filter has the following 5x5 kernel:

$$\begin{bmatrix} 0 & -0.5 & -0.5 & -0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & -0.5 & -0.5 & 0 \end{bmatrix}$$

In addition to the horizontal smoothing that the weak filter accomplishes, it also enhances horizontal edges. Therefore it is preferable to the horizontal filter in situations where contrast is poor. The strong filter tends to produce more features than the weak filter. Therefore, if the strong filter is applied and the number of features produced is larger than a set critical value (e.g., 400), the program will repeat the filtering and segmenting procedures, replacing the strong filter by the weak filter.

The filter choice is made automatically, based on the amount of contrast in the superwindow and on flow deviation, according to the following table:

	deviation of flow angles	
contrast	low deviation	high deviation
high	weak filter	lowpass 3x3 filter
low	strong filter	none

9.4 Feature descriptions and definitions

After filtering the superwindow is segmented ("thresholded") into black and white regions. The regions correspond to ridges and valleys (respectively) in the original print. The segmentation is based on local (relative) grey levels and also the one dimensional grey level gradient in the vertical direction.

A column by column scan is performed on the binary data, identifying ridges and valleys by building collections of like colored pixels called **features**. A feature is a ridge or valley that is maximal with respect to unique continuation, i.e., a **feature** ends at either a ridge ending or ridge bifurcation. See Fig. 9.2.

Each end of each feature is classified as one of the types **root**, **branch**, **gap**, or **edge**; the first three are displayed in Fig. 9.3, and the last category merely means that the end falls off the edge of the superwindow. Note that these definitions are color independent: a valley (white region) is handled in the exact same manner as a ridge (dark region).

Note that when a superwindow is thresholded and segmented, typically 100 to 200 segments are generated. The program has space allocated for 480 segments. If the segment data structure overflows, something is radically wrong with the

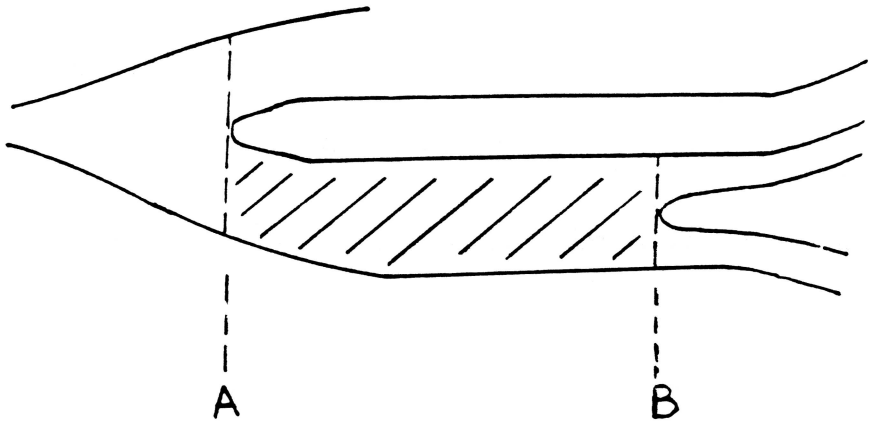


Figure 9.2: Schematic explanation of feature definition. Feature starts at cross section A as a branch and ends at cross section B as a root.

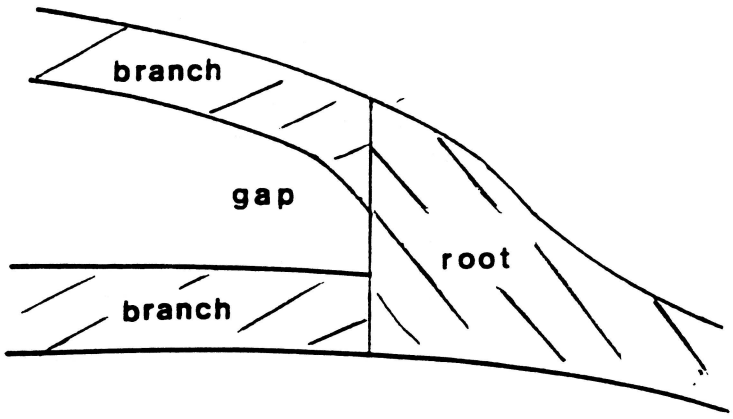


Figure 9.3: Schematic showing the definition of feature end classification.

quality of the fingerprint in that superwindow. If this occurs, then the region is considered to be of poor quality and no minutiae are extracted from it.

9.5 Feature selection under noise

9.5.1 Feature preprocessing

After the feature structure is built it is necessary to do some processing before minutia extraction. In particular, small (false) holes, gaps, branches, and bridges are removed (refer to are Fig. 9.4). Next the edges of the features are smoothed and made square.

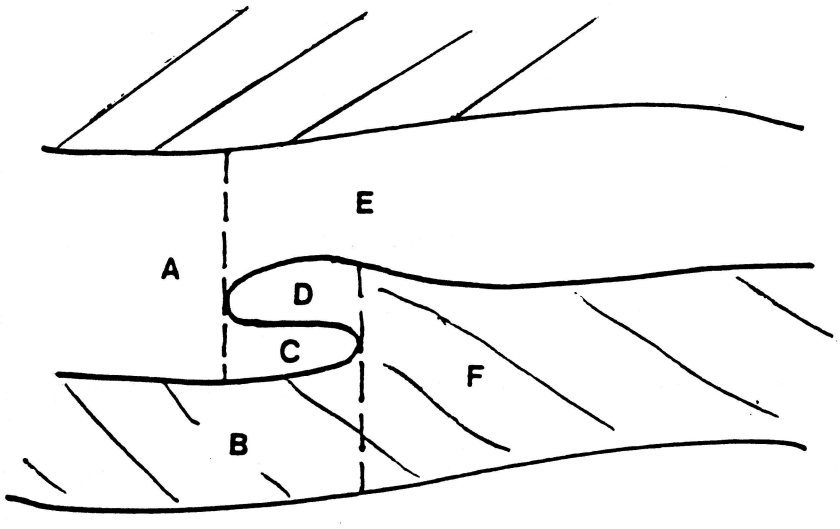


Figure 9.4: Schematic of false branches, indicated by letters C and D. They are declared false since they are "short". One will be removed by program.

9.5.2 Minutia classification

The two main types of minutiae are ridge bifurcations and ridge endings. Refer to Fig. 9.3. If one considers the branch and root to be dark (i.e., a ridge) and the gap to be light (i.e., a valley), then this pattern represents a ridge bifurcation. On the other hand, if one considers the gap to be dark and the branch/root to be light, then it represents a ridge ending. This observation is the basis for our minutia extraction procedure. Instead of distinguishing between ridge bifurcation and ridge endings, we look only for gap type endings. We place a minutia at each gap type end. The code to do this is independent of the gap color. After minutia placement the gap color can be considered and the distinction between ridge ending and ridge bifurcation can be made. We have determined, however, that this is not a useful distinction because it is not stable. If a finger is reprinted then due to dirt or uneven inking of the finger a branch of a bifurcation can break off and appear on the new print as a ridge ending (and of course the bifurcation has disappeared). Likewise, a ridge ending can be falsely joined to a neighboring ridge and thus appear as a bifurcation. Therefore, we make no distinction between ridge endings and ridge bifurcations.

Instead of ending versus bifurcation distinction, we make a distinction based on "minutia direction". See Fig. 9.5. For each gap define the minutia direction to be the flow direction oriented to point from the gap into the corresponding root. Note that unlike the flow direction, which is defined in the range $[0,\pi)$, the minutia

direction lies in the range (-π,π]. In terms of ridge bifurcation and endings this definition has the following interpretation: Ridge bifurcations take the orientation that points into the bifurcation, and ridge endings take the orientation that points away from the ending. Notice that this definition is stable with respect to the ridge-to-bifurcation and bifurcation-to-ridge mutations discussed above.

9.5.3 Defect classification and identification

Each gap type end in the current subregion is a potential minutia. However, due to dirt, pores, or breaks in the print or due to artifacts of the segmentation and processing routines, it may happen that the gap is a defect (i.e., a false minutia). Defects occur in pairs, most of which can be classified into three main types with two subdivisions for each type. Refer to Fig. 9.6.

Type 0 defects are illustrated in Fig. 9.6 (a) and (b). The forward type is a false break in a ridge and the backward type is a hole. Note the marked corresponding gaps. For the forward type the feature between the gaps has a root type end at both the left and the right ends. The marked gaps are the gaps for each root. Thus we consider the gaps to be at the same "level"—hence the classification as Type 0 (i.e., the difference in "levels" is zero). Likewise for the backward type the gaps are gap ends at opposite ends of the same feature. Again the same level and again classified as Type 0.

Fig. 9.6 (c) and (d) show the standard Type 1 defects, forward and backward,

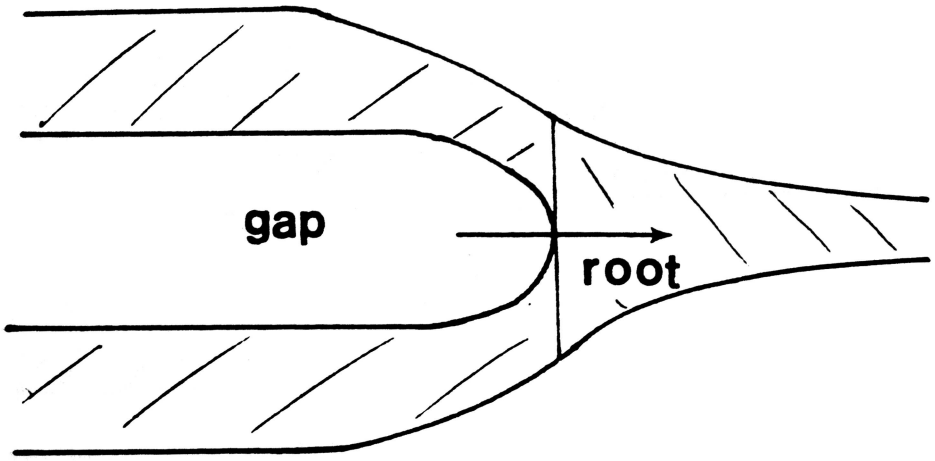


Figure 9.5: Schematic showing the definition of minutia direction

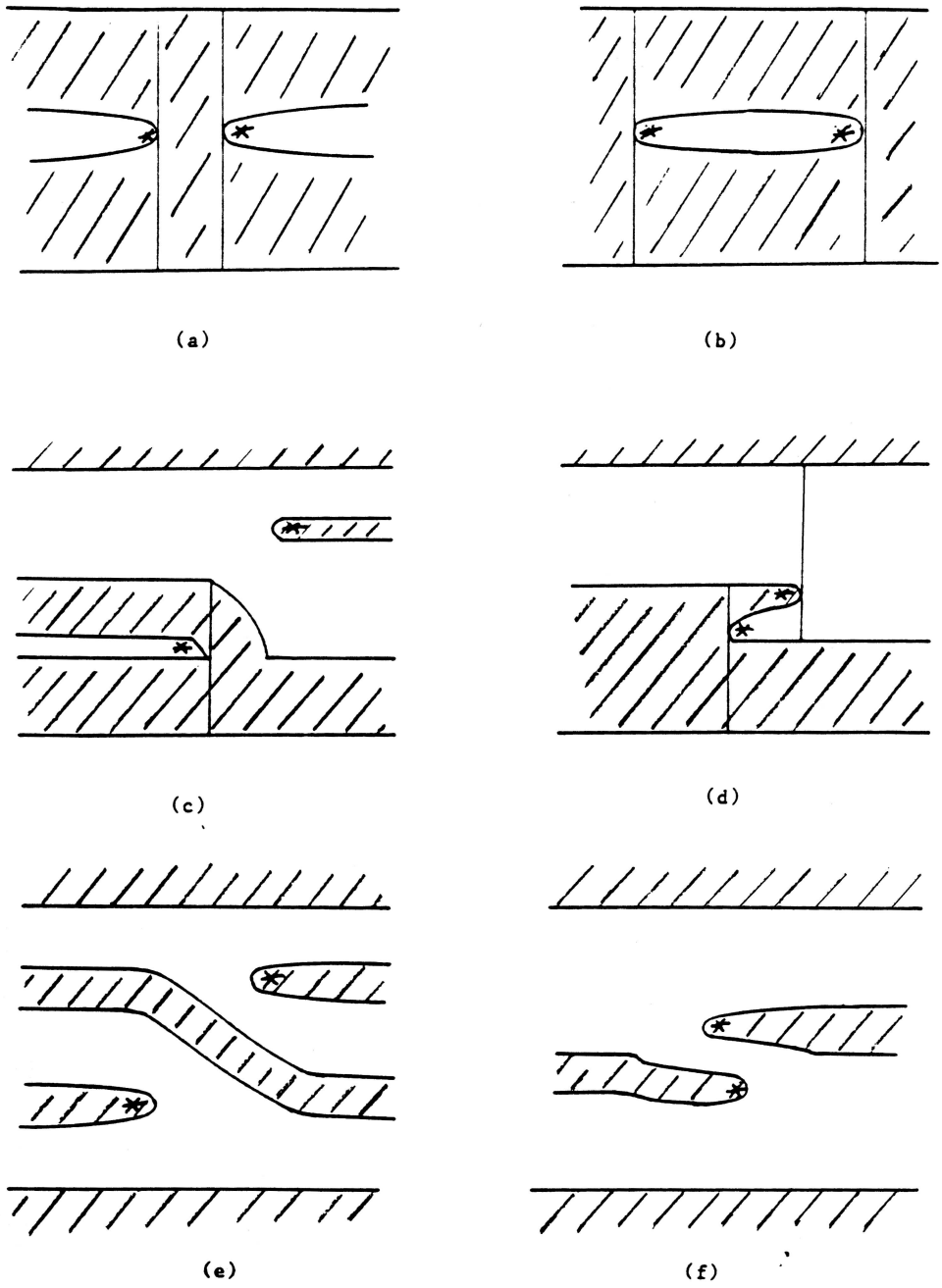


Figure 9.6: Schematic classification of feature defects. Gap ends are marked by stars. (a) Type 0, forward. (b) Type 0, backward. (c) Type 1, forward. (d) Type 1, backward. (e) Type 2, forward. (f) Type 2, backward.

respectively. Consider the forward type first. Notice that the root for the left hand gap is immediately below the root for the right hand gap. Thus a difference of one level and hence the classification as Type 1. This type of defect is characteristic of a false ridge break followed by a false joining of one end of the break to an adjoining ridge. Next consider the Type 1 backward defect. Here the feature to which the left hand gap belongs is immediately below the feature to which the right hand gap belongs. Again a difference of one level and hence a Type 1 defect. Compare to Fig. 9.4. This is exactly the false branch type that is one of the false features that the **remove_false_features()** routine attempts to remove. At this latter stage in processing, however, larger and more complicated defects can be handled than for which the **remove_false_features()** routine is designed. Note also that for Type 1 defects the corresponding gaps are of opposite colors, whereas for Type 0 defects they are of the same color. The general rule is that even typed defects (Type 0, 2, etc.) have corresponding gaps the same color, while odd typed defects (Type 1, 3, etc.) have corresponding gaps of opposite color.

The rule for the subtype depends on the minutia direction. If the minutia on the left has direction toward the right (and then the minutia on the right will have its direction toward the left), then the subtype is forward (one looks in the 'forward' direction from either minutia to find the corresponding minutia). The backward subtype is defined similarly. The general rule for counting levels thus depends on the subtype. If the subtype is 'forward', then look at the corresponding roots for

each gap and count the number of features difference straight up or down between them. For the backward type, work with the feature to which each gap belongs. Using these rules the reader should have no trouble verifying the classifications for Fig. 9.6 (e) and (f). Higher order type defects (Type 3, 4, etc.) could be defined but at the current stage of development only Types 0, 1, and 2 are used.

The assignment of a gap type end into one of the above defect classes depends on many factors. The corresponding gap must exist and must be fairly close, and also the neighboring ridge geometry must be appropriate. As an example, consider the Type 0 forward defect (false break) of Fig. 9.7. The probability depends directly on the ratio of distance v_2 with respect to the sum $v_1 + v_3$ and depends inversely on the distances h_1 and v_4.

If a gap end is not classified as one of the above defects, then a check is made to see if the local ridge geometry is appropriate for a true minutia. In particular, a check is made to see if the features on either side bend together. If the local geometry is inappropriate, the minutia is classified as being in a bad region. Alternately the minutia is considered to be a true end.

Actually for each gap end a probability is computed for each category (true end, Type 0 defect, Type 1 defect, Type 2 defect, and bad region), and the gap end is classified according to the category with highest probability. If the gap is classified as a true end, then the gap coordinates are stored as a minutia location.

If a gap is classified as a Type 0, 1, or 2 defect, then a corresponding gap has

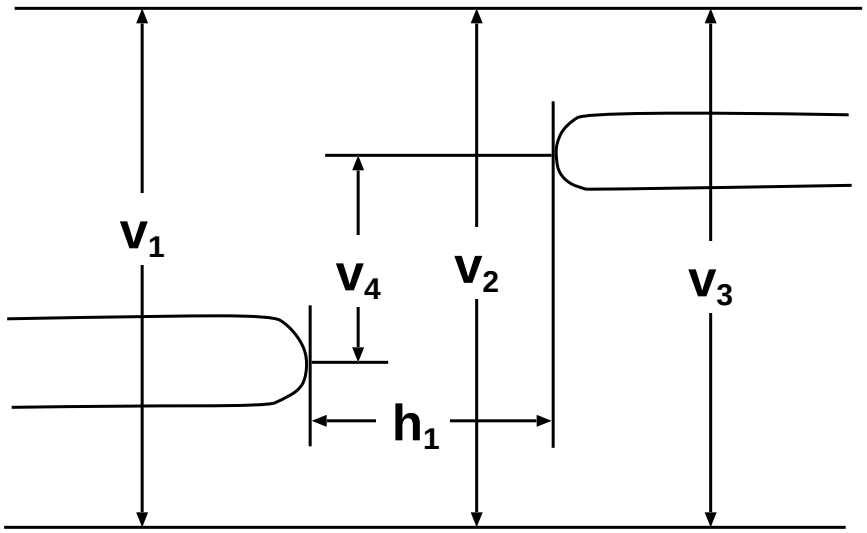


Figure 9.7: Illustration of measurements used by algorithm to define Type 0 forward defect probability.

been identified and theoretically the defect may be removed (repaired). Ideally, therefore, the defect classification routine should be run iteratively to make tentative classifications and those gaps that have high probability of being of Type 0, 1, or 2 should be repaired after each step. This iteration has yet to be implemented.

9.6 Feature extraction

The minutiae from all the subregions are collected together in the structure **extracted_minutiae**. Recall that the subregions are designed to overlap, so minutiae in the overlap area should be listed more than once. The location can be shifted somewhat due to inconsistencies in the segmentation routine and due to round off errors. Thus each minutia in an overlap region is compared to other nearby minutiae to identify repeats. If two minutiae in an overlap region are closer together than approximately one ridge pair width (defined below) with minutia directions closer together than 45°, then the two are identified as one and the location and minutia direction for the minutia is taken as the average.

It may also happen that a subregion of a good section of the print may by chance have no minutiae in it. This does not mean that there is no information to be extracted from the subregion. There is still local topology information such as flow, curvature, and ridge width. This information is generated on a regularly spaced mesh called the topology grid. Stored at each mesh point are the angle and curvature scores (derived straight from the curvature grid), a probability field,

and a new field which is the average ridge pair width. This last field is determined from the feature structure interpretation of the superwindow, and is defined to be the average width (in rows) of a ridge/valley pair. It was decided to work with pair widths as opposed to ridge (or valley) width since the latter depends on ridge/valley thresholding (which at present is not completely reliable). The probability field is a measure of the reliability of information in that area. It is based on the curvature uncertainty (the probability is inversely proportional to curvature/flow uncertainty), but is modified also by the ridge pair width stability. In particular, if the ridge pair width is found to vary greatly over the small local region, then it is likely that any information derived from this region is unreliable (most likely because the region is of poor quality), and so the probability is reduced.

The flow charts for the upper level extraction routines are given in Fig. 9.8 through 9.12.

9.7 Pseudocode for the top several functions of the program

The material below assumes the entire fingerprint is to be processed, i.e., **fp_sp**="f".

9.7.1 Pseudocode for the main routine

Note that the main routine does nothing but call two routines and take care of some distracting details. Since our policy in writing pseudocode is to omit distracting details, our pseudocode for the main routine has only two statements. The major

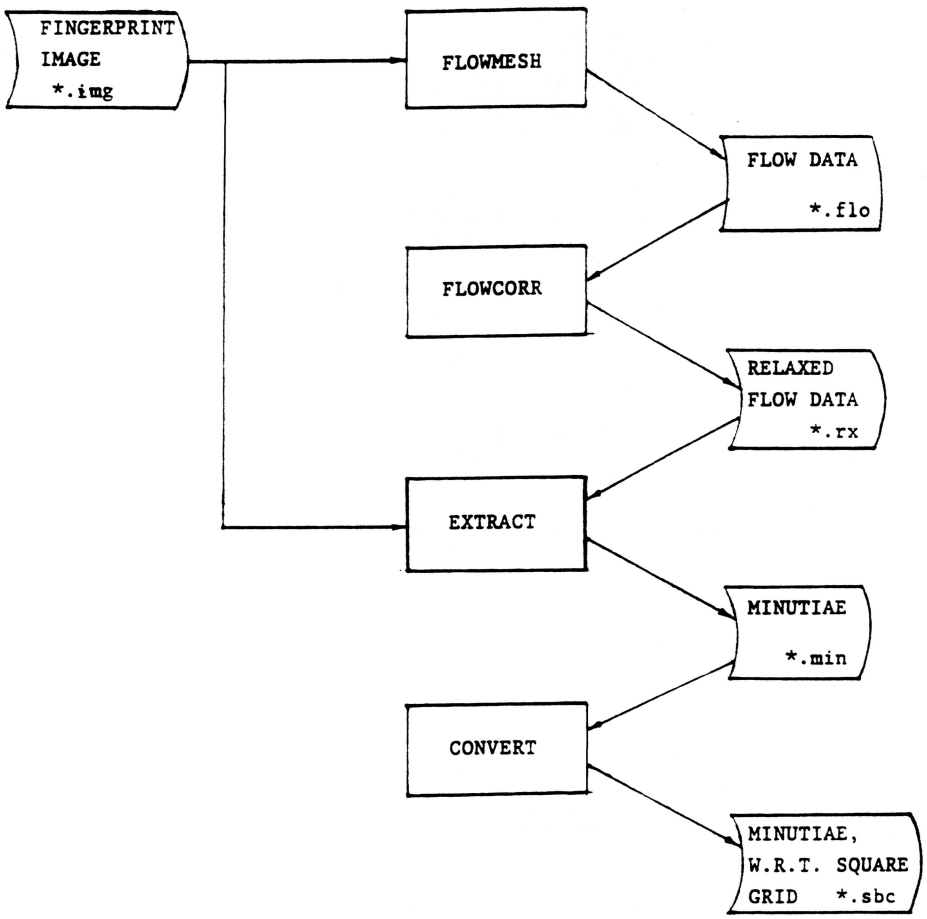


Figure 9.8: System flow chart for fingerprint latent processing.

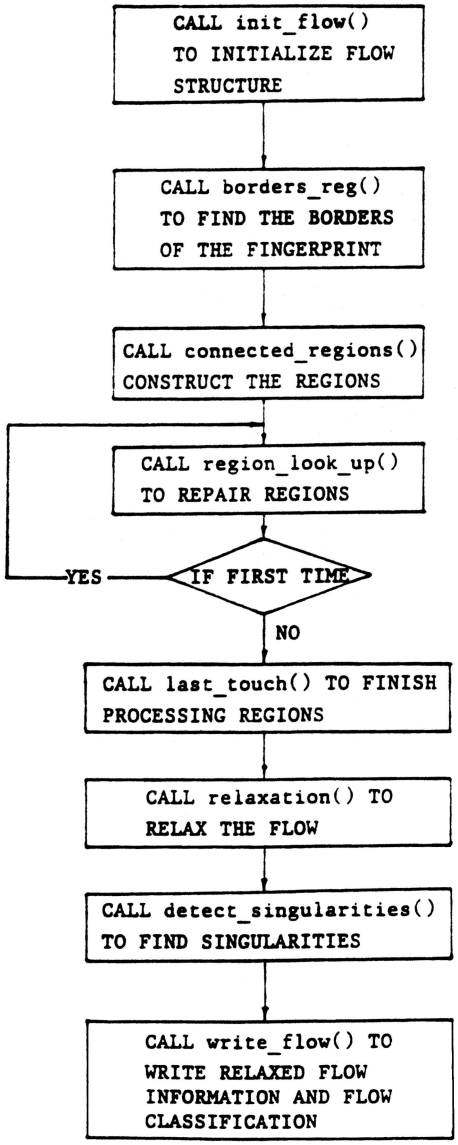


Figure 9.9: Flow chart for mainline of flow correction program.

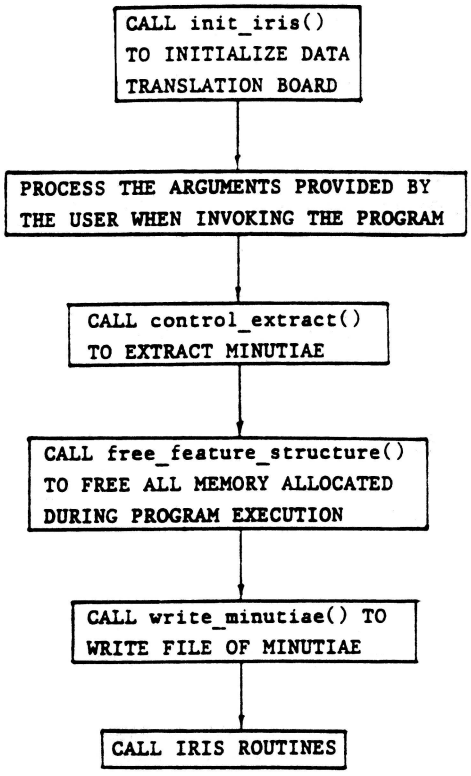


Figure 9.10: Flow chart of mainline for program extract.

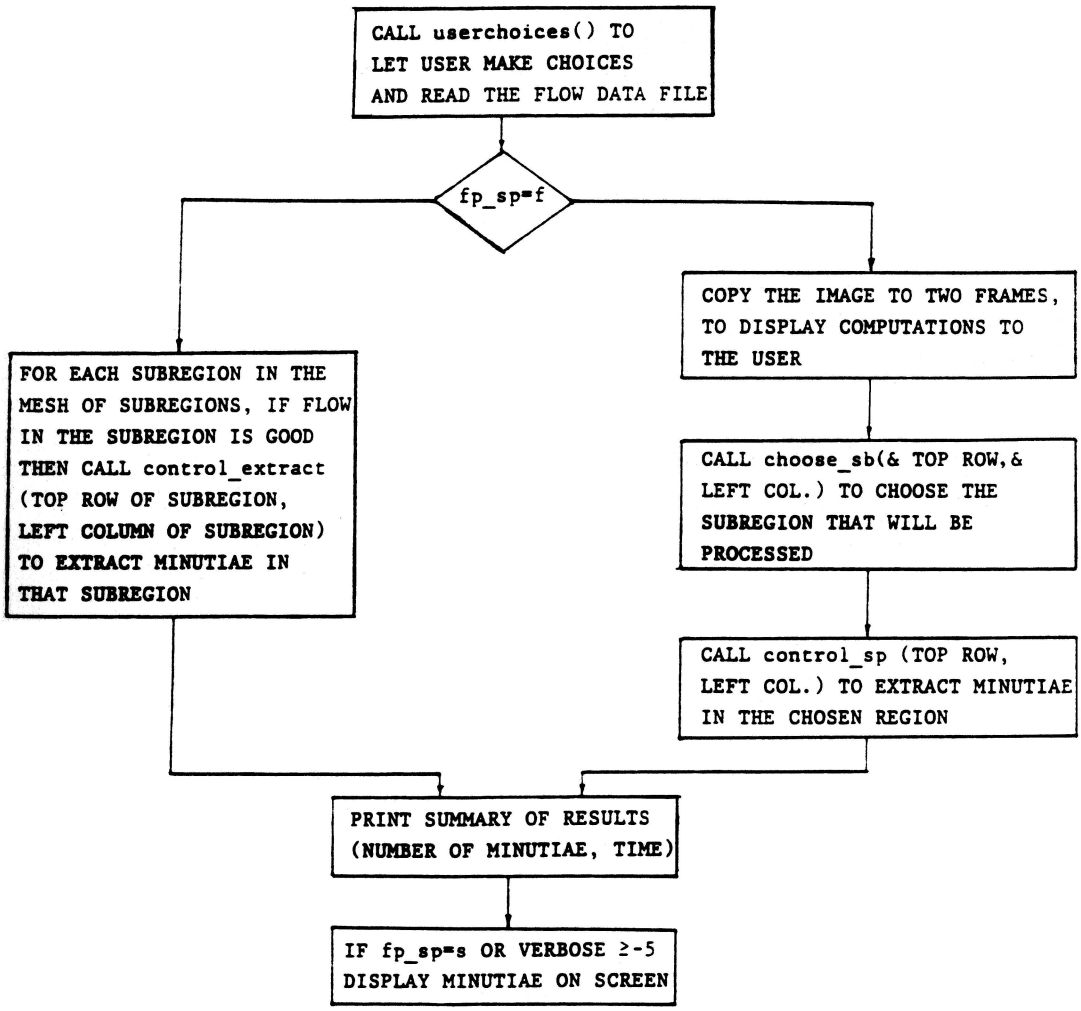


Figure 9.11: Flow chart of subroutine control_extract in program extract.

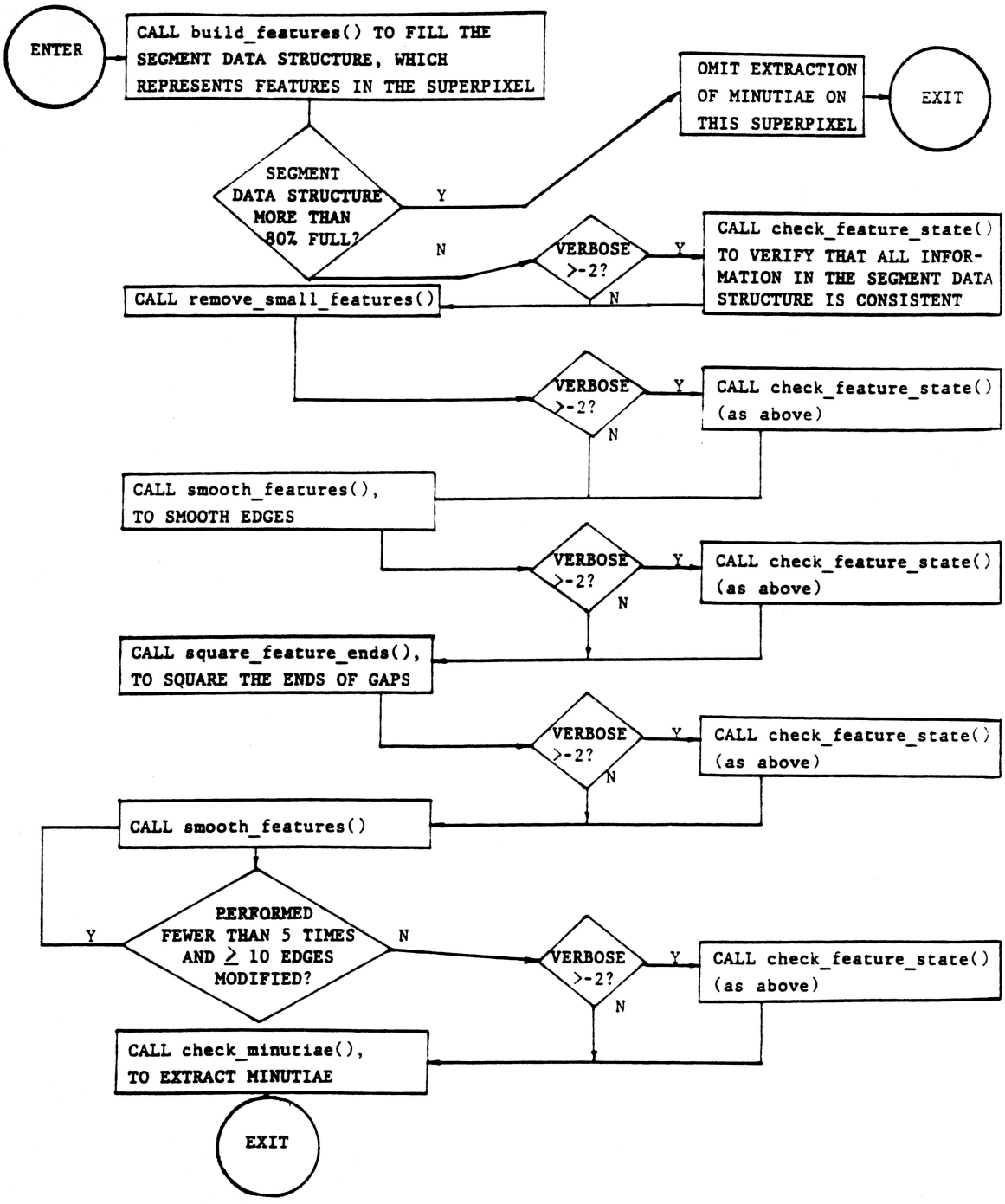


Figure 9.12: Flow chart for subroutine process_sp in program extract.

detail taken care of in the main routine is the processing of arguments that the user provided when invoking the program. (See the section on running the extraction program.)

1. Call **control_extract()** to control all extraction that will be performed.

2. Call **write_minutiae()** to write a file of all minutiae found during extraction.

9.7.2 Pseudocode for control_sp()

1. Decide whether the subregion will be processed as a single standard sized subregion, or split into 4 quadrants. The decision is based on flow deviation.

2. For this one subregion, or for the 4 quadrants it was split into in Step 1, and for each rotational angle at which it must be processed:

   (a) Call **rotate_and_fill_sp()** to define a superwindow around the subregion, download the superwindow from the image, rotate it the selected amount, and store it in the global array **superwindow**.

   (b) Call **fill_sp_flow()** to fill the global array **sp_flow**, which contains flow values (relative to the selected rotation) in the superwindow.

   (c) Filter the superwindow using a strong filter, weak filter, or a low-pass 3x3 filter (or none at all). The filter is applied by calling **filter_superwindow()**.

(d) Call **segment_sp()** to threshold the superwindow. Until thresholding, each pixel has intensity level between 0 and 255; in the thresholded image each pixel is either black or white. The result is stored in the global character array **thresh_sp** (with values B and W).

(e) Call **process_sp()** to clean up the superwindow and extract minutiae on it.

(f) If a strong filter was used and too many features were created, then go to 2c for the purpose of trying again, using a weak filter this time.

(g) Call **report_virtual_minutiae** to calculate average ridge pair width and to store topological parameters (curvature, width, and probability) in the **vir_min** array.

9.7.3 Pseudocode for process_sp()

1. Build the segment data structure, to represent the features in the superwindow.

2. Remove small features (by calling **remove_small_features()**). All bridges, all holes, and some small branches are removed. A feature is a **bridge** if all of the following are true: (1) both ends are roots, (2) its length is less than its width, (3) it is shorter than the gaps it abuts, and (4) the direction from the left gap to the right gap is not too different from the flow direction. A feature is a **hole** if both ends are gaps, and it is short or very thin. A feature

is a **small branch** if one end is a branch, the other end is a gap or falls off the edge of the superwindow, and the feature is small or narrow. A small branch is removed unless some neighboring feature is a skinnier branch.

3. Smooth edges (by calling **smooth_features()**).

4. Call **square_feature_ends()**, to shorten long skinny ends of gap-type features.

5. Repeatedly smooth edges (by calling **smooth_features()**). Smoothing is performed at most 5 times; quit earlier if a call to **smooth_features()** changes fewer than 10 edges.

6. Extract minutiae (by calling **check_minutiae()**).

CHAPTER X

Image identification through feature matching

This chapter covers feature matching aspects and algorithms developed by the author in two distinct contexts: defect detection in printed circuit boards and latent fingerprint identification.

In printed circuit board defect detection features represent wire trace ends and bifurcations. Defects manifest themselves as new, absent, or modified features. Morphological variations in the test board relative to the reference board are insignificant, and the image quality is good, so all differences in observed features are of importance. This is in stark contrast to latent fingerprint identification. Here features from a latent print (lifted, for example, from a crime scene) representing ridge endings and bifurcations need to be compared against a large library of features from inked prints for identity determination. Generally the features from the latent print represent only a tiny fraction of the total features found on the inked print. Dirt and smearing and low quality samples in general introduce false features and obscure true ones. Moreover, the elasticity of skin can introduce

significant morphological changes to a print.

Feature extraction methods for these two situations were described in previous chapters. The present chapter is a natural continuation and conclusion to that work, presenting not only matching algorithms but also results that depend not only upon the matching algorithms but through them upon the quality of the feature extractions. The simpler printed circuit board problem, considered first, is from a coauthored paper [1], where the feature matching material presented in this chapter was contributed by this author.

10.1 Defect identification on printed circuit boards

After extracting the feature points from the board under test, a comparison between the test board features and the reference board features is necessary. This comparison requires two phases: alignment and matching.

10.1.1 Alignment

Alignment marks are normally placed far apart from each other on a board, so as to minimize alignment errors. A two dimensional coordinate system is introduced to the reference board such that the first alignment point of the reference board has coordinates $(0, 0)$ and the second alignment point lies along the positive x-axis. The board under test has a coordinate system introduced implicitly from the feature extraction process. The goal of the alignment phase is to convert the implicit coordinate system of the test board into the coordinate system that has

been imposed on the reference board.

Suppose the test board has N features at locations $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, with (x_1, y_1) the coordinates of the first alignment point and (x_2, y_2) the coordinates of the second alignment point. These coordinates are translated to bring the first alignment point to the origin and then are rotated about the origin to bring the second alignment point to the positive x-axis. The equations to do this are

$$\begin{aligned} x'_i &= x_i - x_1 \\ y'_i &= y_i - y_1 \qquad\qquad i = 1, 2, \ldots, N \end{aligned}$$

and

$$\begin{aligned} x''_i &= (x'_i x'_2 + y'_i y'_2)/(x'_2 x'_2 + y'_2 y'_2)^{1/2} \\ y''_i &= (y'_i x'_2 - x'_i y'_2)/(x'_2 x'_2 + y'_2 y'_2)^{1/2} \qquad\qquad i = 1, 2, \ldots, N, \end{aligned}$$

where $(x'_1, y'_1), \ldots (x'_N, y'_N)$ are intermediate results and $(x''_1, y''_1), \ldots, (x''_N, y''_N)$ are the final results. Notice that $(x''_1, y''_1) = (0, 0)$ and that $x''_2 > 0$ with $y''_2 = 0$.

10.1.2 Matching

The goal of the matching phase is to determine if the features of the board under test correspond in a one-to-one fashion with the features from the reference board.

Let (a_1, a_2) be the coordinates of a feature from the reference board, and let (b_1, b_2) be the coordinates of a feature from the board under test. These two features are said to match if they are the same type of feature and if the distance

```
Use alignment points of test and reference boards to align boards.
For each feature point A of test board:
    For each unmatched feature point B of reference board:
        If A and B are the same type and distance between A and B is
        less than fitting tolerance ε:
            begin
                Match A and B.
                Break (i.e., proceed to next feature point on test board).
            end
```

Figure 10.1: Algorithm for comparison of test board features against reference board.

between them is less than a given fitting tolerance ϵ (i.e., $\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} < \epsilon$).

Now for each feature on the test board an attempt is made to find a match with a feature of the same type from the reference board. If such a match can be made, and if the feature from the reference board has not been previously matched to another feature from the same test board, then both the feature from the test board and the feature from the reference board are recorded as matched. Note that this procedure assures that no two features of the test board get matched to one feature of the reference board or vice versa. Pseudo code for this algorithm is displayed in Fig. 10.1.

It is important that each feature from the test board not have the opportunity to match more than one feature from the reference board. If this condition is violated then incorrect pairings may result. For example, refer to Fig. 10.2. In

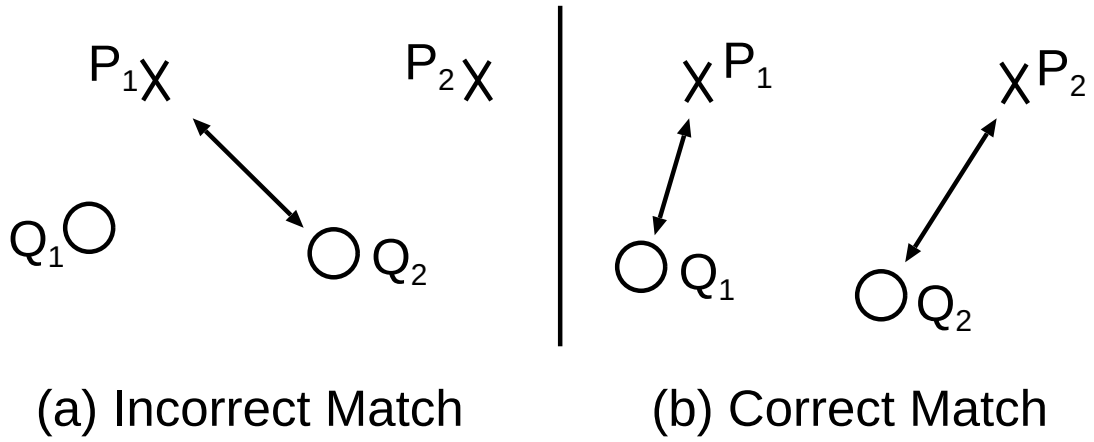


Figure 10.2: Matching of reference features P_1, P_2 with test board features Q_1, Q_2.

this example P_1, P_2 are features from the reference board, and Q_1, Q_2 are features from the test board. Feature Q_2 has the option of matching to either P_1 or P_2. If it were decided to match Q_2 to P_1, as on the left, then P_2 and Q_1 would be left unmatched. However, if it were decided to match Q_2 to P_2, as on the right, then all the points get matched.

This problem occurs when the fitting tolerance ϵ is too large. To avoid this difficulty, ϵ should be chosen smaller than half the minimum distance between pairs of points on the reference board. If this is done then it is impossible for a point from the test board to be closer than ϵ to two points on the reference board. Note that this condition depends only on the location of features on the reference board and is independent of events on the board under test.

Another aspect of the matching algorithm is the procedure by which prospective

candidates for matching are brought together. That is, given a feature point from the board under test, how does one locate a feature point from the reference board to match with it? The simplest method is to check against every feature point of the reference board. But this procedure can be very time consuming. For example, suppose the reference board and the test board have 1000 feature points each. Then for each feature point from the test board, 1000 comparisons must be made. Hence the total number of comparisons for the whole test board is 1,000,000 comparisons.

A better approach is to divide the reference board into rectangular regions as indicated in Fig. 10.3. For each subregion a list is made of the feature points of the reference board that lie in that subregion. Then, given a feature from the test board, one determines which subregions are close enough to that point so as to possibly contain a match candidate. Hence, only these few subregions, each with a very short list of features, need to be checked for matches (the actual number of subregions that must be checked depends on the size of the regions, the location of the test feature point with respect to the subregion boundaries, and the fitting tolerance). Under ideal conditions each subregion will contain on the average about one reference feature point, so the number of comparisons can be shaved to close to the number of test feature points. For instance, in the previous example with 1000 feature points on both boards, the number of comparisons may be on the order of only several thousand comparisons. This represents a considerable time

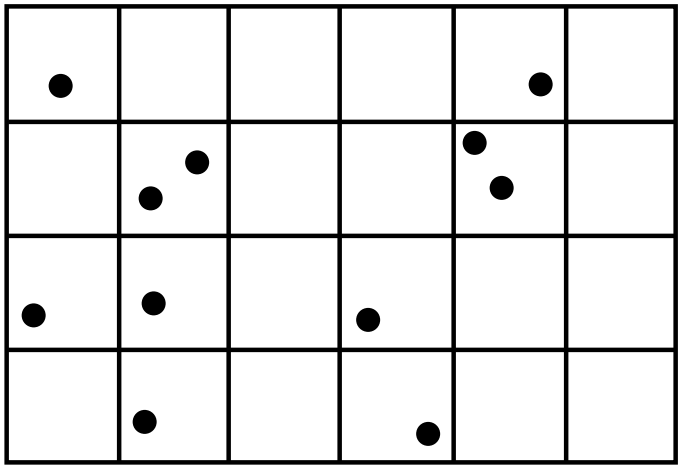


Figure 10.3: Alternative approach to feature point organization for comparison, using 2-dimensional grid cells.

savings. Other more sophisticated data structures and algorithms for this problem are described in [69, Chp. 5].

10.1.3 Output

The output from the comparison routine is a list of features missing from the test board (features unmatched from the reference board) and a list of false features on the test board (features unmatched from the test board). From this output it is possible to identify the source of failure.

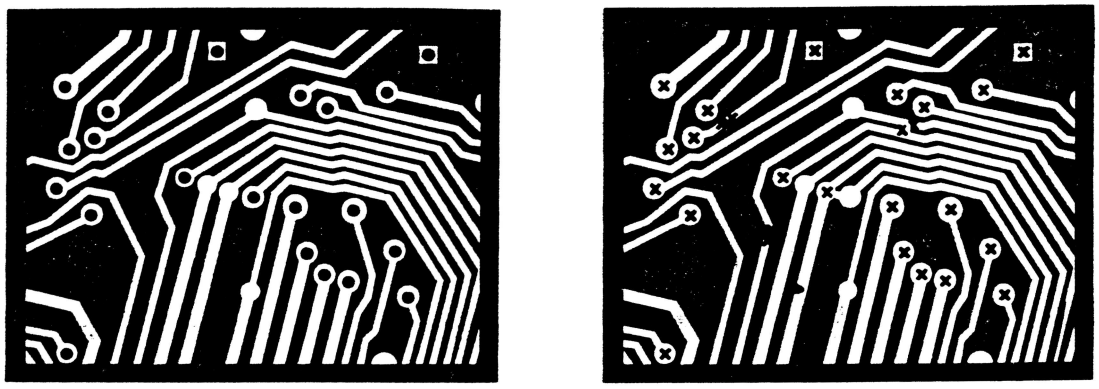
10.1.4 Generalizations of Comparison Algorithm

The above comparison algorithm can be generalized to meet requirements arising from other applications. For example, consider an application in which alignment

points could not be easily introduced to the objects that are to be compared. In this environment the alignment phase of the comparison is considerably more difficult.

One solution is to arbitrarily choose a pair of features from the reference board to be used as the alignment points. Then an attempt is made to find a pair of features on the test board that are the same type and the same (approximate) distance apart as the distinguished pair from the reference board. Suppose such a pair can be found. Then treat this pair as alignment points and go through the alignment/matching procedure. Repeat for each candidate feature pair from the test board, and choose the results with the fewest errors as the outcome of the comparison. This procedure may also be repeated by consecutively choosing several pairs of features from the reference board to be used as alignment points, and attempting to compare the test board with each.

Another generalization of the comparison algorithm is to step away from the rigid condition of requiring one-to-one correspondence. For printed circuit boards, if the features from the test board do not match exactly with those from the reference board, then the test board is faulty. But in other applications a partial match may be satisfactory. In these cases it would be necessary to judge the quality of a partial match.



(a) (b)

Figure 10.4: (a) Section of reference board with solid circles marking extracted features. (b) Section of test board with crosses marking extracted features, including defects.

10.1.5 Experimental results

In our experimental system images are obtained using a standard video camera. The video signal is digitized using a Data Translation IBM-AT compatible arithmetic frame grabber. This board is also used for data transfer and data representation on a monitor. The digitizing and processing boards are housed in a 33 MHz Intel 386 based computer with numeric coprocessor. The algorithm was realized using the C programming language and compiled under version 5.0 of the Microsoft compiler. Processing for the image displayed in Fig. 10.4—determining feature points and finding design rule errors—takes slightly under 13 seconds. This image has 190,000 pixels, and occupies just over 3 square inches on the PCB.

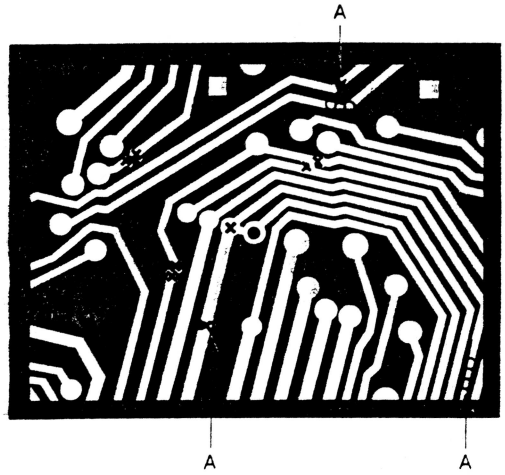


Figure 10.5: Unmatched feature points (from both boards) and wire width violations (marked by arrows A) overlaid on the test board image.

The output of the comparison algorithm is a list of those features on each board that were left unmatched. Fig. 10.5 displays this output overlaid on an image of the test board together with design-rule violations marked. In Fig. 10.5 the crosses represent unmatched features from the test board, and the solid circle is an unmatched feature from the reference board. The empty circles are design rule errors. The actual representation on the computer screen is colorized and may be easily interpreted by the operator.

10.2 Latent fingerprint identification

For the purposes of matching, not all the information in a fingerprint image is needed. So a subset of this information, consisting of minutiae and topological

data, is extracted. This extracted information is what is referred to in this paper as 'fingerprint data'. We are interested in two such sets of data: the 'latent print data' and the 'file print data'. The latent print is the unknown print (coming for example from a crime scene), and the file print is one from some library of known prints, to which we want to compare (match) the latent to determine if they are from the same individual.

For reasons of efficiency the match is broken down into two stages. For each file print a set of possible orientations for the latent (relative to the file print) are considered. Each orientation is subjected to a screening test that has as its purpose the quick exclusion of obviously incompatible prints. If an orientation passes the screening test, then it is subjected to a more thorough examination.

10.2.1 Stage 1: Screening match

In the first stage of the matching algorithm, one minutia point is chosen from the latent print and is designated as 'the match point'. Then a subset of the latent in the neighborhood of the match point is used to form a 'screening latent'. The size of this screening set is currently 8 minutiae. Fix a point in the file print; let us call it the candidate point. The screening latent is translated so that the match point overlaps the candidate point, and then the screening latent is rotated so that the minutia direction of the match point is the same as the minutia direction of the candidate point. Consider the screening latent to be 'overlaid' on top of the

file print. If the two prints are from the same individual, and if this is the proper relative orientation, then the minutiae from the screening latent should overlap minutiae from the file print. If half of the points in the screening latent "overlap" points from the file print then we have a candidate orientation (for the latent print relative to the file print) and the match passes to the second stage (the second stage of the match will be described below). This screening is done for all points in the file print, i.e., EACH point in the file print generates a candidate orientation. Each candidate orientation that passes the screening gets sent to the second stage match. If no point in the file print generates a candidate orientation that passes the screening test, then the file print is decided to not match the latent. Note that passing the screening test is a necessary condition for the latent to be considered to match the file print, but it is not sufficient. Also it is possible that a single file print will generate several candidate orientations that pass the screening test.

Before we describe the second stage of the match let us define what is meant by minutiae "overlapping". Of course some deformation and minutiae misplacement is inevitable. As a result the latent points cannot be expected to line up exactly with minutiae from the file print. For the first stage match two tolerance parameters, ϵ_1 (set to 26 pixels, i.e., approximately 2.5 ridge pair widths) and θ_1 (set to $\pi/6$), are used to allow for this distortion. Let L be a minutia point from the rotated/translated screening latent print, and let F be a minutia point from the file print. Note that a minutia point consists of a location and a minutia angle.

In order for L and F to be considered as overlapped, it is required that the distance between their locations be less than ϵ_1 and that the difference between their minutia angles be less than θ_1. Otherwise the minutiae L and F are considered as non-overlapping.

10.2.2 Stage 2: Detailed comparison

Note that the screening match described above uses only minutiae data. In the second match stage we use both minutiae and topological information. The total match score (measure of how closely the file and latent correspond) is broken down into two components: the minutiae score and the topological score.

10.2.2.1 Minutiae score

The minutiae scoring of the second stage of the matching is conceptually similar to the screening test of the first stage. The main difference in the second stage is that now the comparison is made using the entire latent minutia set instead of the smaller screening latent set. The second difference is more subtle. Instead of merely counting "overlapping" points, in the second stage we determine an "overlap score".

If a minutia from the rotated/translated latent has the exact same location and minutia angle as a point from the file print, then the overlap score is maximized. As the distance between the locations and the difference in minutia angles grows,

the overlap score decays until it reaches zero. The base tolerance parameters for this scoring, which where determined experimentally, are ϵ_2 (set to 16 pixels) and θ_2 (set to $\pi/16$).

Given a minutia point from the latent print, a minutia point from the file point is considered to be a candidate match to the latent minutia if the distance, d between the latent point and the file point is less than ϵ_2 and the difference between their directions, denoted by θ_δ, is less than θ_2. The score given to such a candidate match is

$$\text{Match score} = \frac{6}{5}\left(\frac{5\epsilon_2^2}{\epsilon_2^2 + 4d^2} - 1\right) * P_L * P_F + \left(0.5 - 2\theta_\delta^2\right), \tag{10.1}$$

where P_L and P_F are the minutia probabilities for the latent minutia and the file minutia, respectively. There is an additional bonus score of $2 * P_L * P_F$ added if $d < 7$ pixels and $\theta_\delta < \pi/24$. The overlap score for a latent point is the highest match score achieved across all candidate matches from the file print.

The overlap score is computed for each latent minutia point and the results are summed. This total is the minutia match score for the given orientation.

In addition to the above (positive) constituent of the overlap score there are also possible (negative) penalties. In particular, although we expect many minutia points in the file print to be missing from the latent print (the latent print after all is in general only a small piece of the file print and is also likely to be very noisy), the converse is less likely. Therefore, a penalty of $8 * (P_L - 0.5)$ (valid latent

minutia have $P_L > 0.5$) is assessed for each minutia point in the latent that has no minutia partner in the file print that it "overlaps" (i.e., for which the above positive constituent of the overlap score is non-zero). Also, recall that the feature extraction algorithm yields ridge flow direction throughout the file print. In the case of unmatched minutia the match scored is adjusted by $0.5 - 2 * \theta_\delta^2$, where θ_δ is the difference (in radians) between the latent minutia direction and the flow direction of the corresponding location on the file print. Note that this supplement may be either positive or negative, depending on whether or not the two directions agree.

10.2.2.2 Topology score

The main factors in the topology score are curvature, curvature direction, and ridge width information which are passed through the topology grid. This information can be plotted as surfaces with respect to x–y coordinate position (i.e., curvature, angle, and width values plotted against position). Examples of such surfaces for a sample file print are shown in Fig. 10.6 through 10.8. For a given fingerprint such surfaces are referred to as the print's topology surfaces.

If the latent and file print are registrations from the same finger then we expect the corresponding topology surfaces to be similar. In particular the peaks and valleys should coincide. Of course they will not be exactly identical, so some measure of closeness is needed. As a start we considered the absolute value of

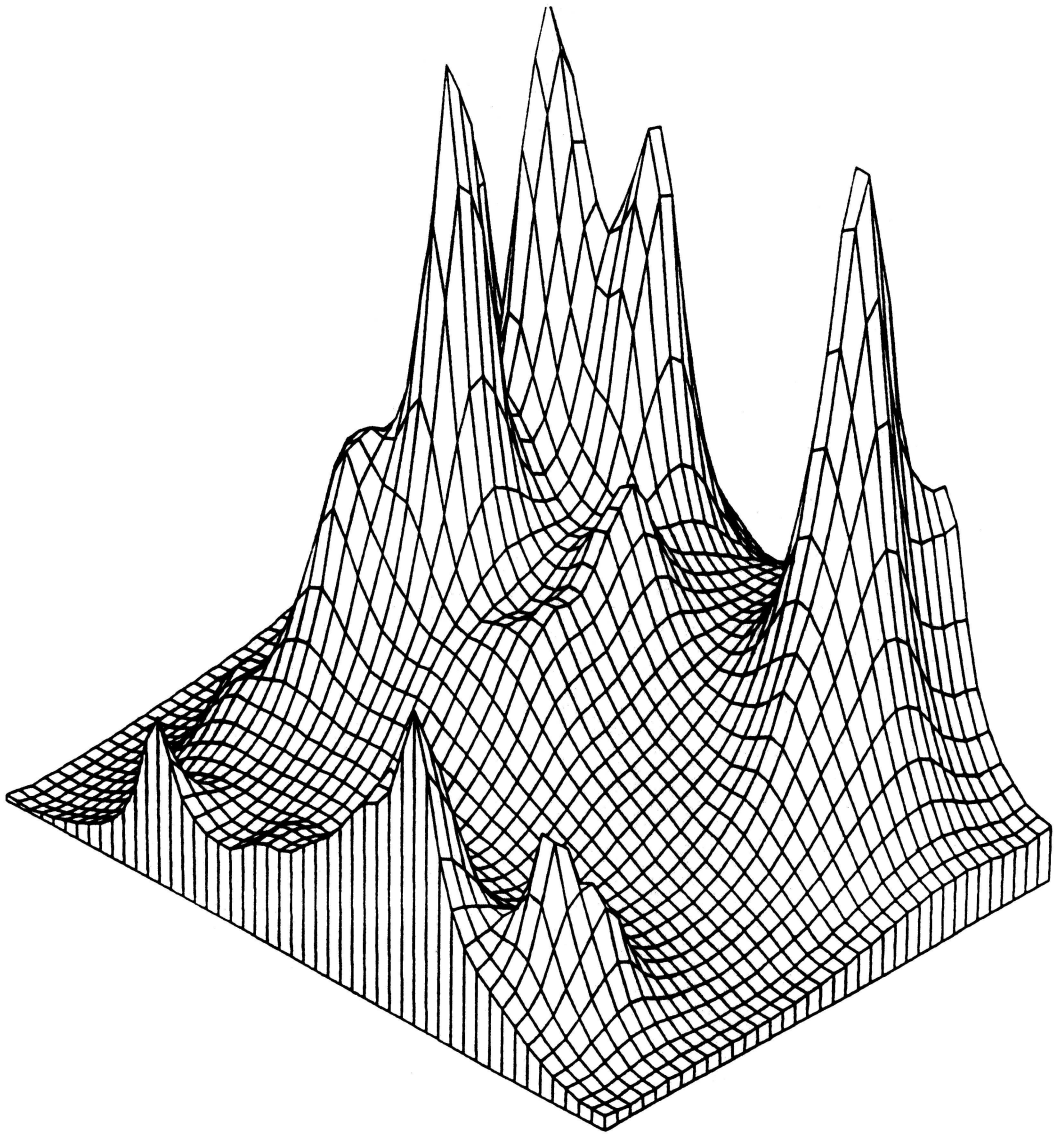


Figure 10.6: File curvature surface.

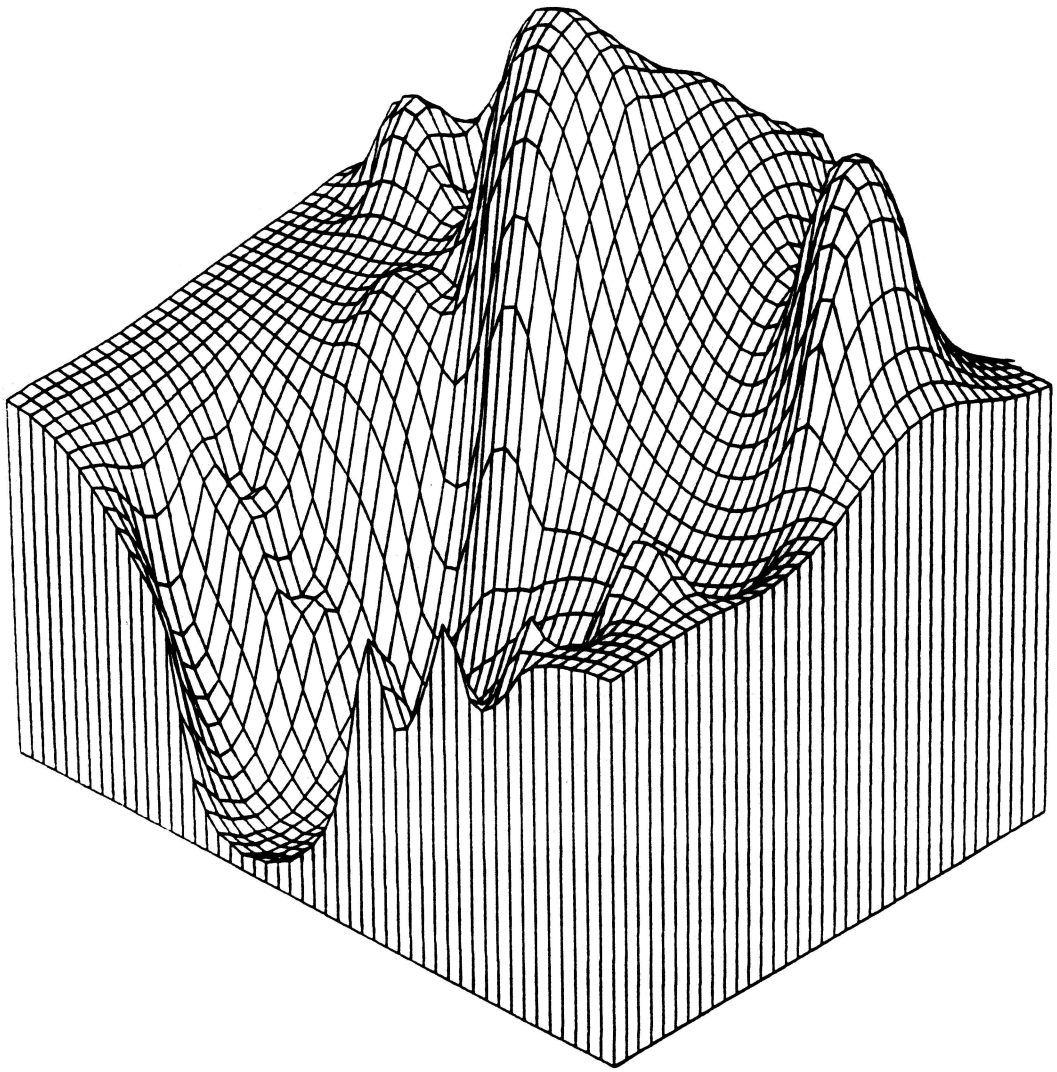


Figure 10.7: File curvature direction surface.

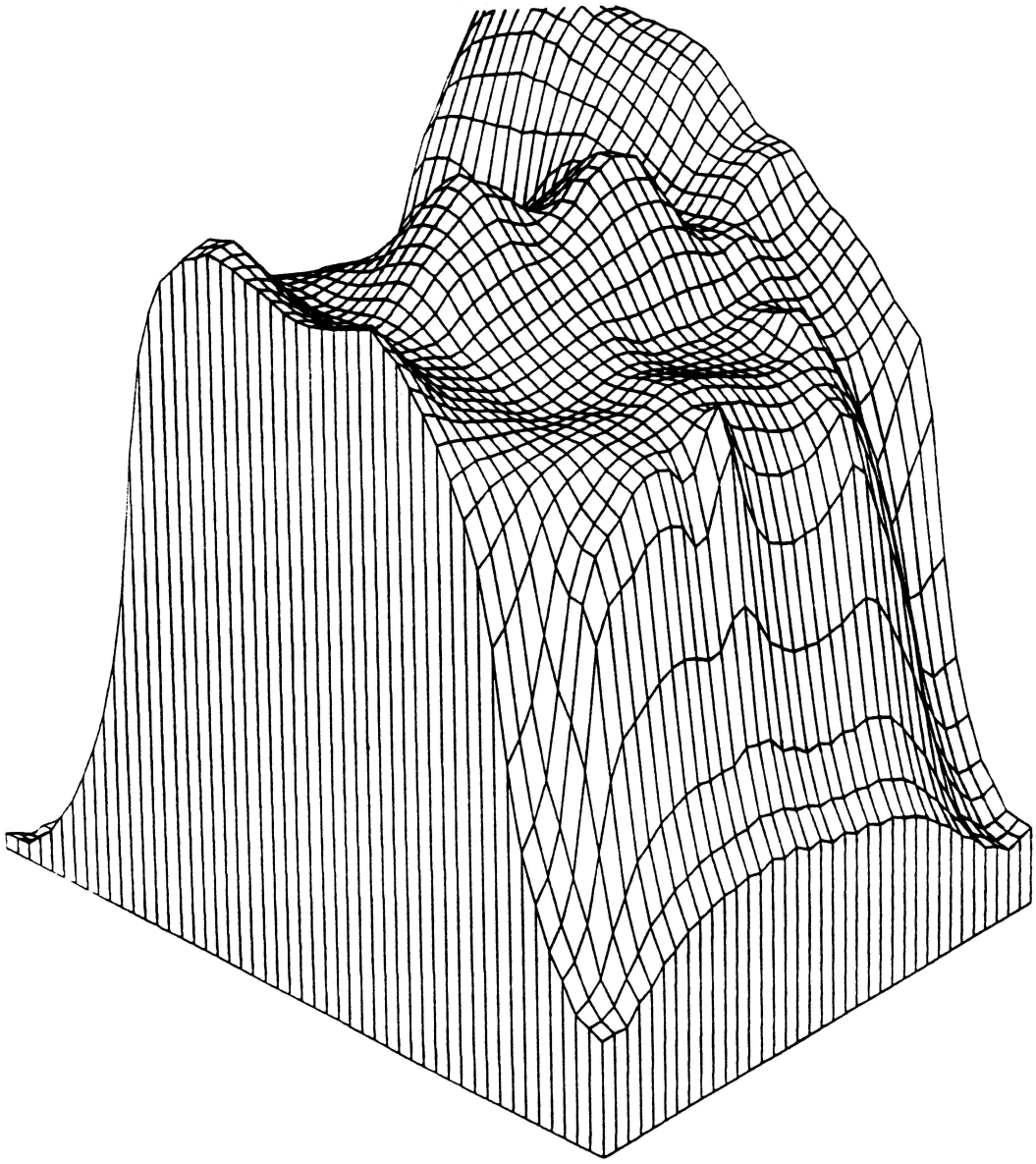


Figure 10.8: File ridge width surface.

the difference between the curvature/direction/width scores at each point of the overlapped topology grids. Clearly, the smaller the differences the better, but with what weight? To answer this question we compared the distributions of these differences for true matches against those for false (non-) matches. The distributions are shown in Fig. 10.9 through 10.11. We discovered that the curvature differences depended heavily on the absolute value of the latent curvature, so the curvature distribution is plotted against two variables—the curvature difference and the latent curvature value.

Consider the range on the direction distribution graph where the true match distribution is above the false match distribution (approximately $\Delta\theta$ between 0 and 0.25). A difference in this range is more likely to result from a true match than from a false match, so each point of the topology grid that yields a difference in this range should contribute positively to the topology match score. Conversely, a difference outside of this range is more likely to result from a false match than a true match, so such a difference should contribute negatively to the topology match score. Graphs of the match scoring functions for each component are shown in Fig. 10.12 through 10.14.

10.2.2.3 Combining the scores

In addition to the minutia score and the topology score two overlap scores are calculated. These overlap scores are a measure of what percentage of the latent

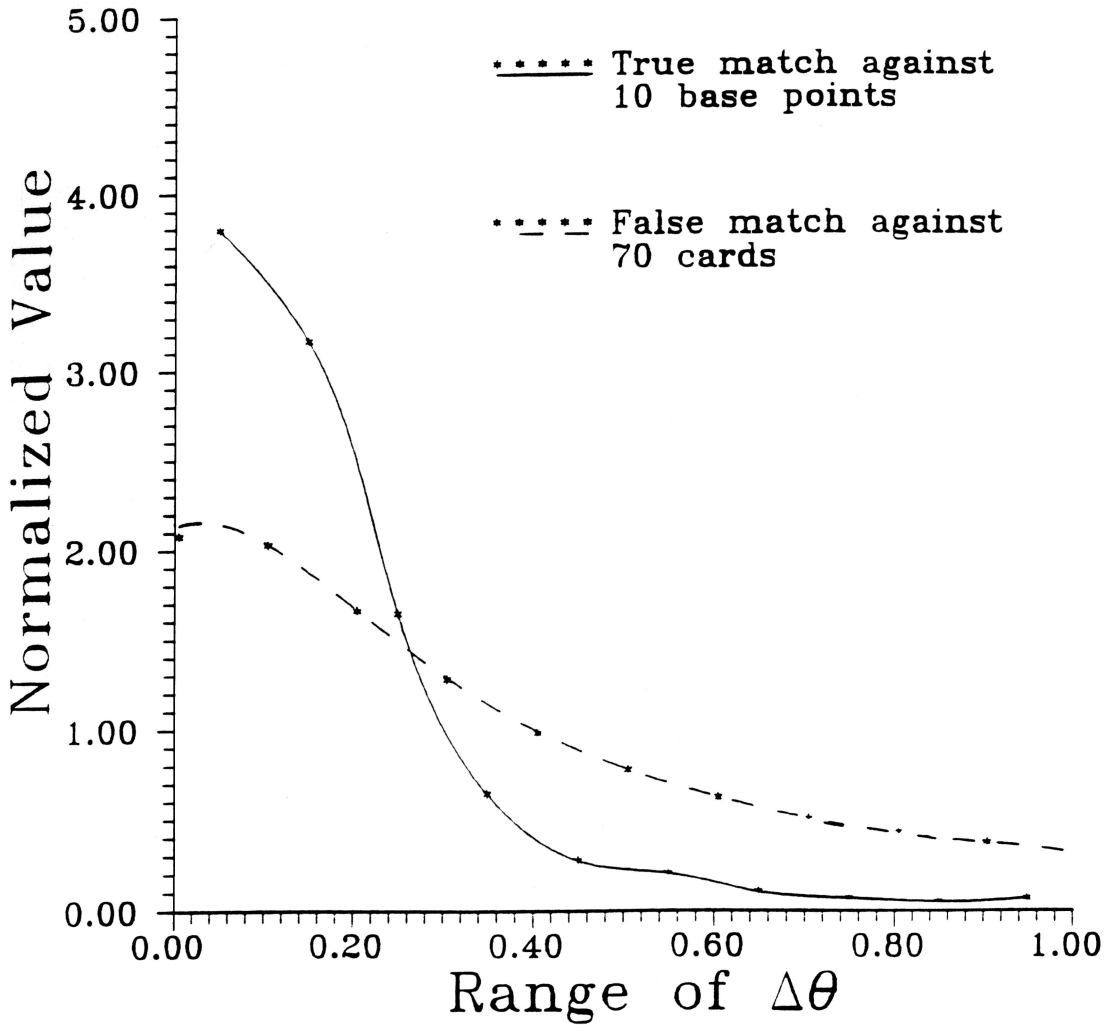


Figure 10.9: Distribution density functions of curvature angle difference for true match (solid) and for false match (dashed).

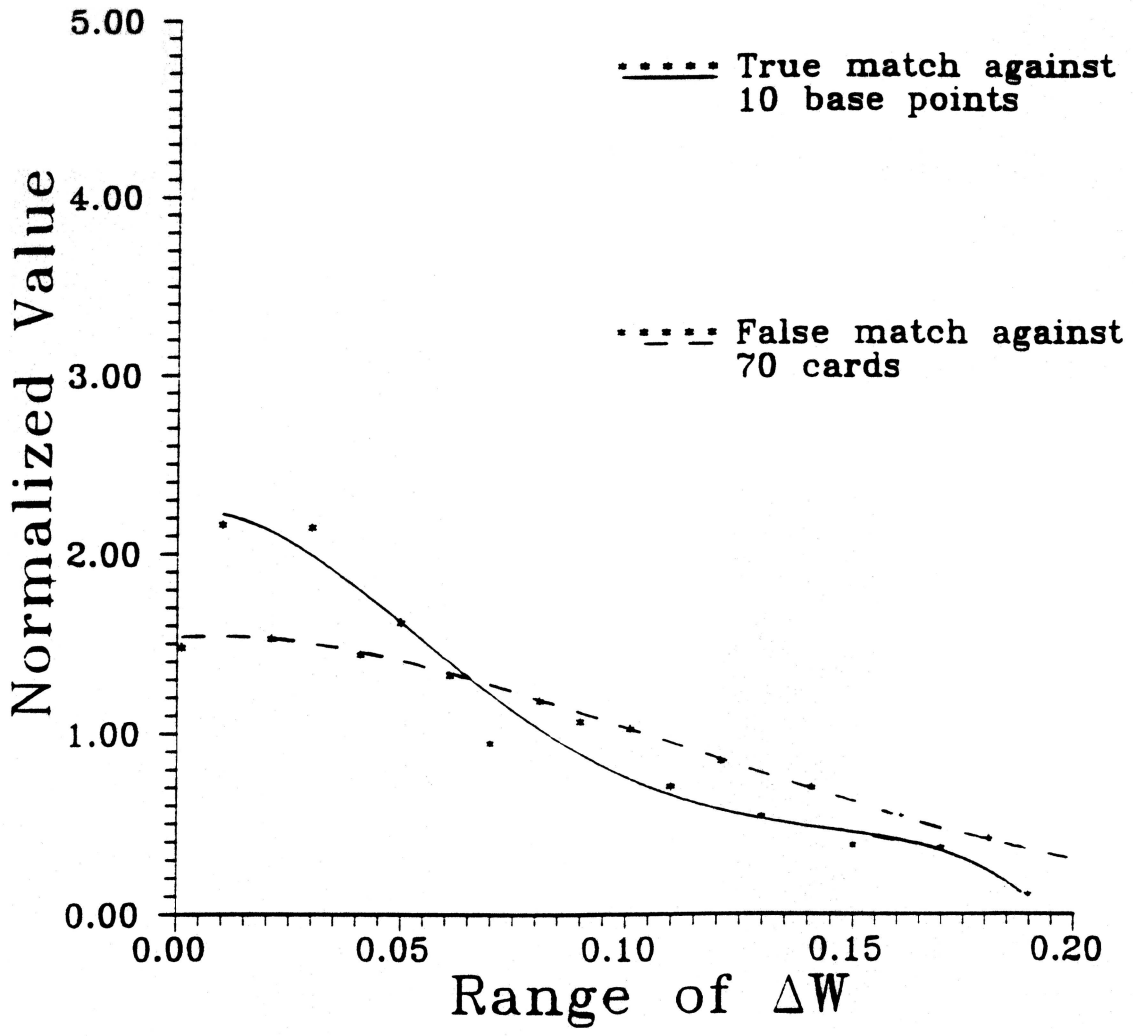


Figure 10.10: Distribution density functions of ridge width difference for true match (solid) and for false match (dashed).

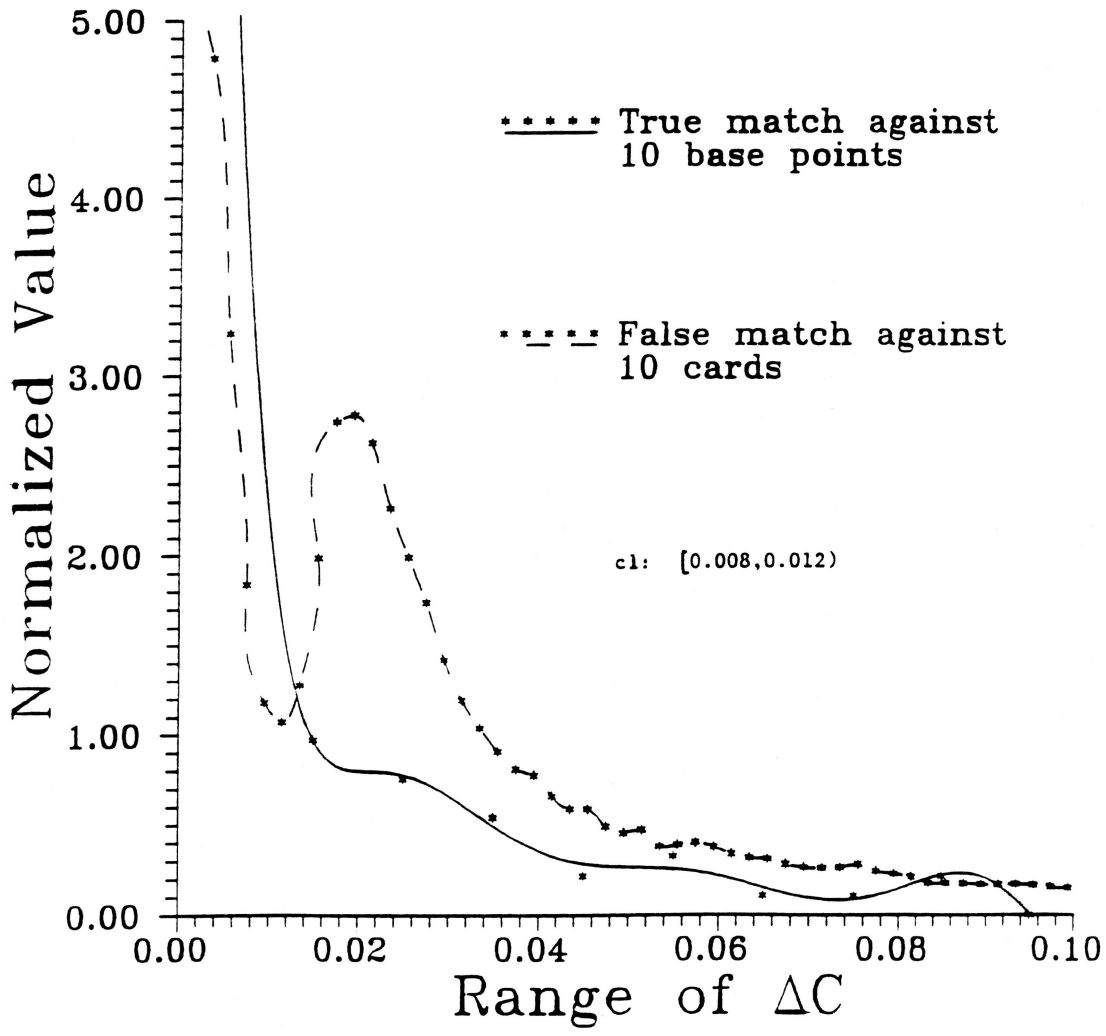


Figure 10.11: Distribution density functions of curvature difference for true match (solid) and for false match (dashed) for latent curvature c_l in the range $[0.008, 0.012)$.

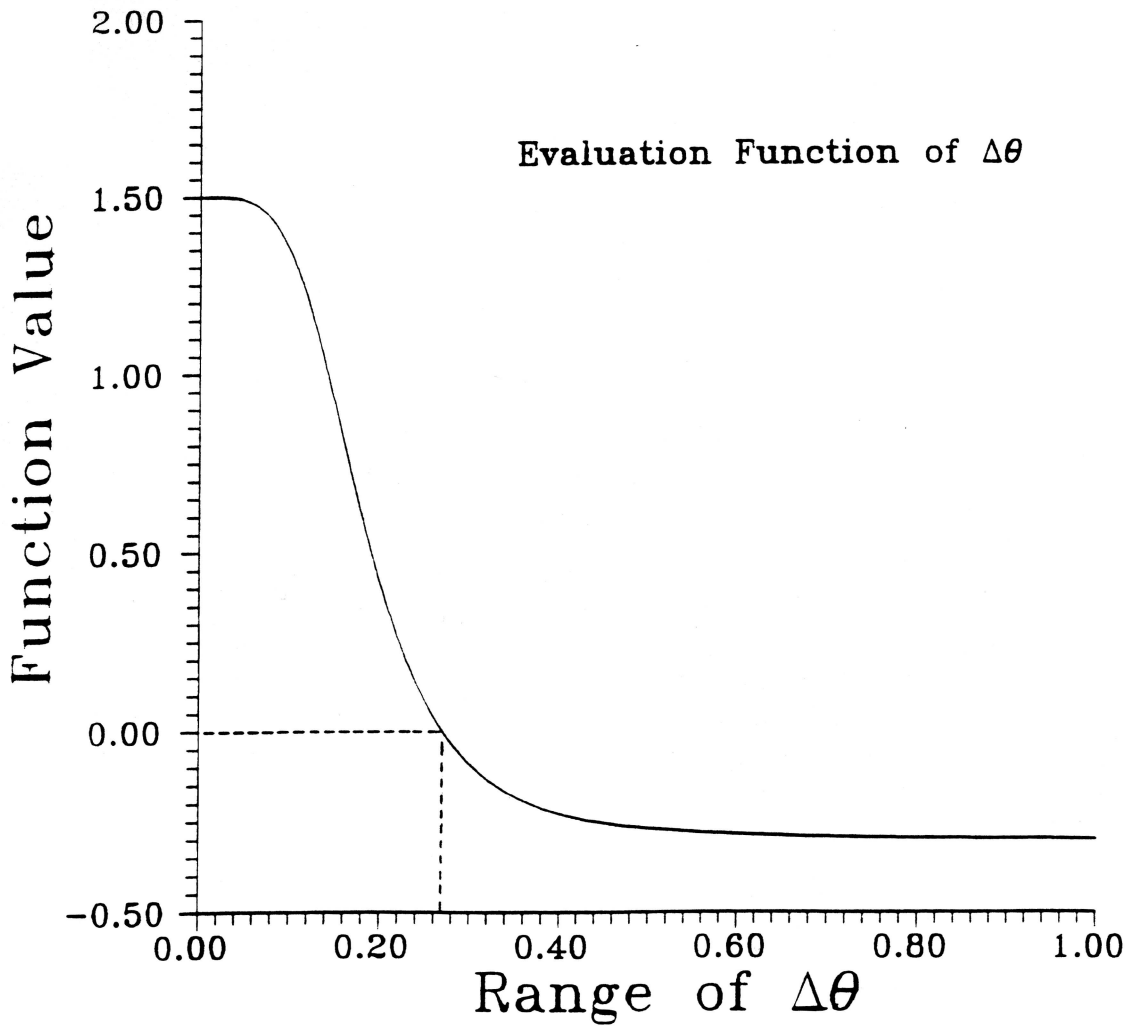


Figure 10.12: Curvature angle difference scoring function.

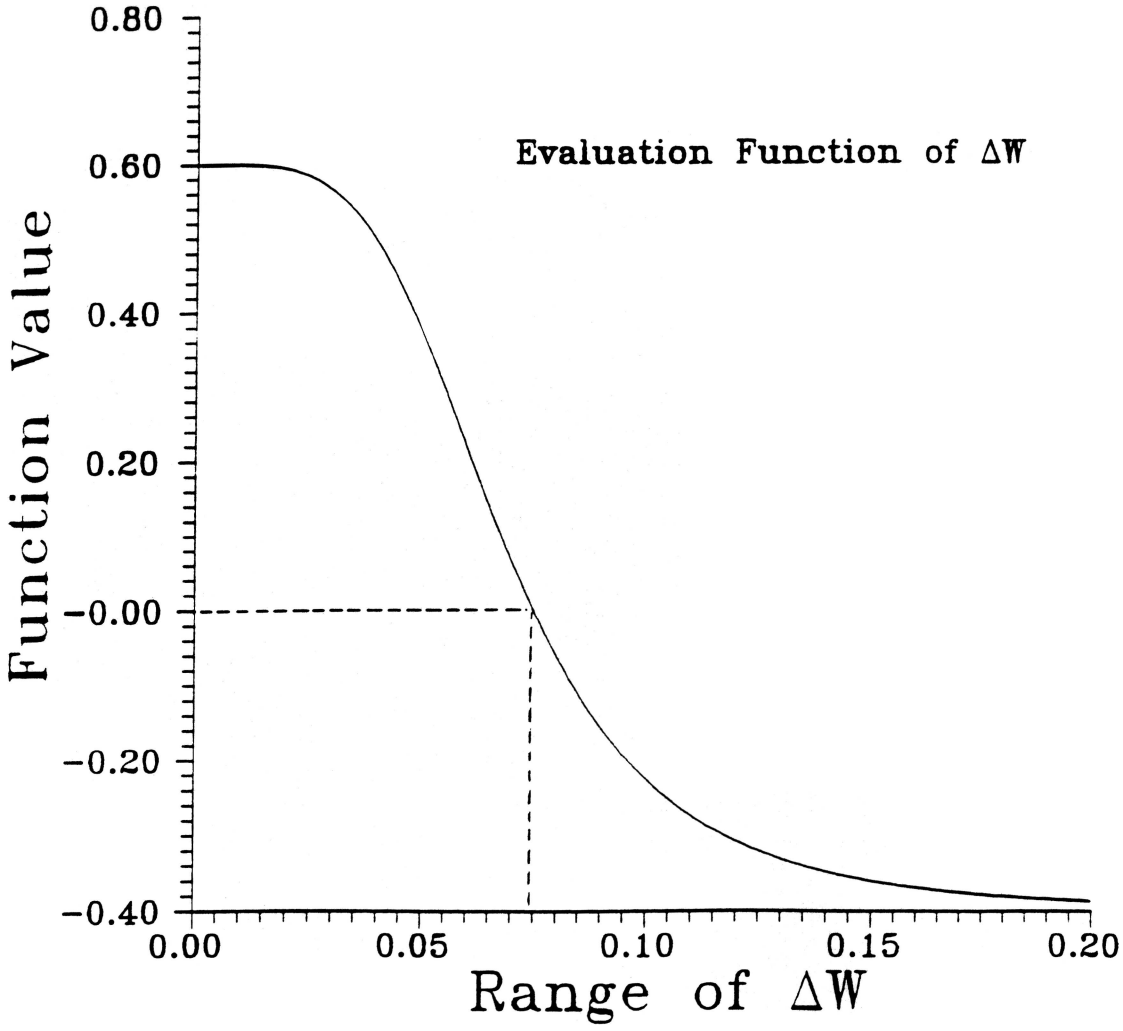


Figure 10.13: Width difference scoring function.

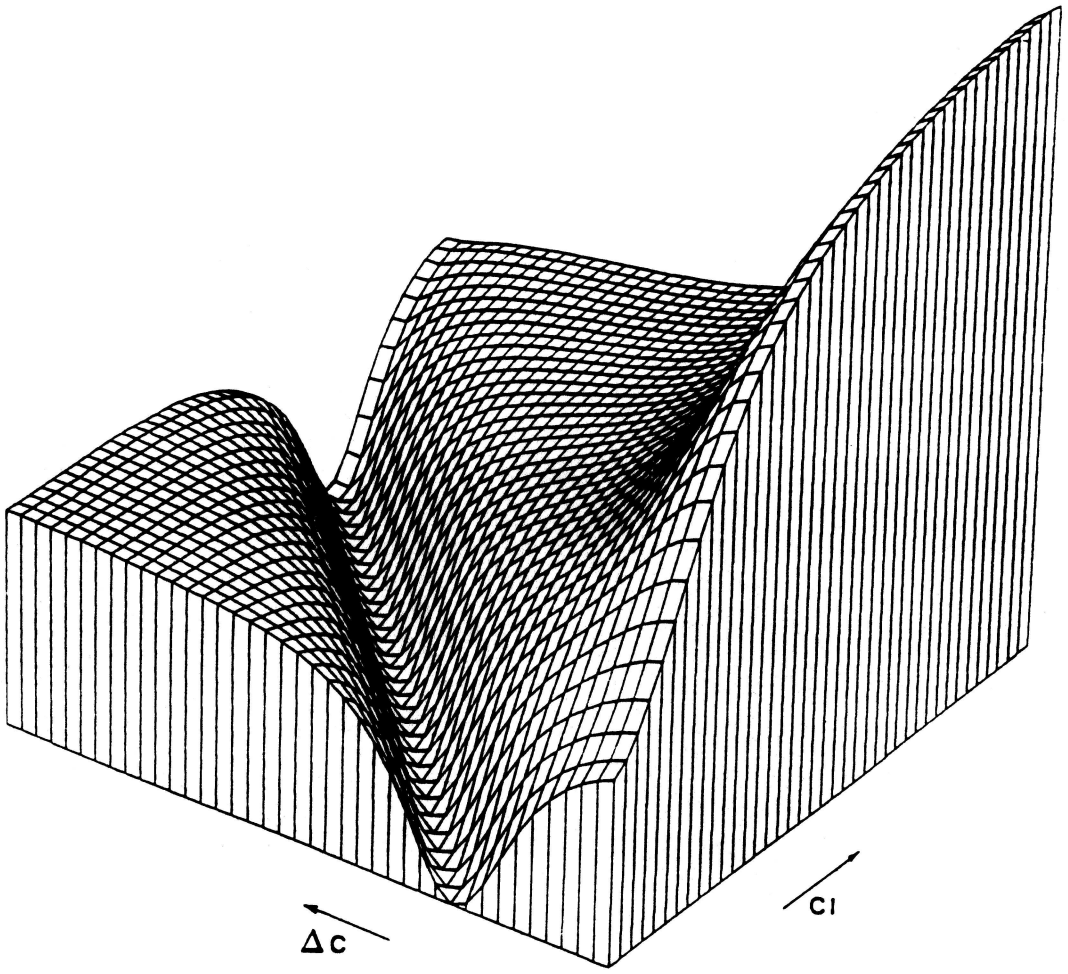


Figure 10.14: Curvature scoring function as a function of curvature difference (Δc) and latent curvature value (c_l).

overlaps the file print. We discovered in the course of our investigation that due to penalties in the minutia and topology scores that a relatively good score could often be obtained by positioning the latent on the file print such that most of the latent was off the edge of the file print. To counter this effect we calculate this percentage and multiply against the topology score. This score is referred to as the TCWR (Theta-Curvature-Width-Ratio) score. This is then multiplied against the total minutia score (the positive minutia score minus the minutia penalty score). This is the match orientation score.

The match second stage is carried out for each candidate orientation that passes the first stage screening. Each candidate orientation generates a match score. The largest match score is considered to be THE match score for the file print. If the latent print and the file print belong to the same individual and finger, then the match score will be high, and if the two prints are from different individuals (or fingers) then the score should be low.

The above algorithm is tolerant to the introduction or deletion of minutiae, as long as the base match point is not affected, i.e., the match point on the latent must be true minutia, and it must also be present in the file print minutia set. To remove this dependency several points from the latent should be chosen as match points, and the above algorithm run for each. Moreover, as will be seen in the results section, true matches usually have high scores for many base match points – a property not shared by false matches. Thus scores from different base match

points can be accumulated and used for true match/false match discrimination.

10.2.3 Deformations in fingerprints

One chief difficulty in automated fingerprint matching is deformation of the print due to the elastic property of skin. This causes the distances between the print minutia to be stretched and/or reduced.

The tolerance parameters used in the minutia score are global in nature—the expected relative distortion between points close together (on one print) is no different than the expected distortion between points far apart. This is too gross a simplification. The topological features are more resistant to deformation, so we are not totally dependent on the minutia score. We can also make the minutia score more robust by breaking the latent into local regions (plates) and performing local matches where we can better control the tolerance parameters. This is a solution that we have partially implemented but have not yet fully studied.

Another way to increase the robustness of the match program is to remove the restriction that the rotated/translated match point "line up" exactly with a point of the file print. Recall that this is how candidate orientations are generated. By allowing in addition small translations and rotations away from exact "line ups," match scores can be made less sensitive to deformations. Currently we optimize the total score against small (less than 15 degrees) rotations. Optimizing against small translations may also be helpful, but is more difficult and we have not yet

attempted this.

10.2.4 Speed of fingerprint matching

Of course the above refinements will increase the time required to compare print minutia sets. Generally it is desired to compare a large collection of file prints to the given latent print, so small match times are required. Current match times are given later in this report along with sample match scores.

One method of decreasing match times is to increase the number of processors working on the task. Currently we have four transputers running concurrently, and increasing this number by a factor of ten or hundred is imaginable. This would decrease match times by close to a corresponding factor. This is a partial solution, which will probably be implemented to some extent. The chief difficulty with this approach is one of communication. Each transputer can directly access only four others, so some 'bucket brigade' method of data communication would have to be implemented.

Another way to increase the speed of the algorithm is to make the minutiae scoring more efficient. The main obstacle in the minutiae scoring is finding those points in the file print that are close to a given location (the location of a point from the rotated/translated latent). Points in a fingerprint have coordinates in 2-space, and storing this information in a manner conducive to an efficient search is difficult. In the original implementation of this algorithm, all minutiae points

in the file print were compared to each point in the latent to determine if any were close. However, in the current implementation, the 512x512 pixel array is broken down into a 52x52 'superpixel' array. The search is then done in two steps: first find the relevant superpixels, and then search only this smaller area. This improvement decreased comparison times by a factor of ten. Note that the local topology information (curvature, flow direction, and ridge width) are on a well defined mesh so that the location of each point is known a priori, so no such problem exists.

It is clear that the second stage of the matching algorithm is more costly time-wise than the screening (first) stage. This is, of course, the purpose of the screening stage. By reducing the number of candidate orientations the speed was increased by a factor of 4 over similar matches without the screening stage. Currently the screening stage uses only minutia position information. If topology information is included in this screening, then it may be possible to increase the speed even more.

10.2.5 Matching results

Results of latent against a 70 card (700 print) library match are collected in Tables 10.1 through 10.5. Consider for example Table 10.1. The original latent image (LCP10) is shown in Fig. 10.15. Five base points were chosen from this latent, and the results for each base point against three different file prints is shown. Fig. 10.16 is the (true) matching file print (MAQ1). Its scores are shown in the first column

of the table. The scores in the first column from left to right are (1) Total (combined) match score, (2) Minutia score, and (3) the topology/overlap ratio score TCWR. The total score is computed via Total = (Min)(TCWR) with negative scores truncated to zero. The average for all 6 chosen match base points is shown in the final row. The other two columns are score results for two false matches. These false match files represent the best scores among the non-matches.

Refer next to Table 10.2. If you refer to the row corresponding to latent base point [237, 293] you will see that for this particular point the false match against file print MAK3 has the best total score. However, if one considers the average across all six base points then MAP6 is the clear (and true) favorite. A similar comments hold for Tables 10.3 and 10.4.

Finally refer to Table 10.5. This is an example of a latent that we were unable to match. Notice however that all the scores against this latent are low. Thus one is not likely to mistake one of the false matches for being a true match. There will always be some latents of such poor quality that they cannot be matched (even by a human fingerprint expert). The results listed here are only for latent prints that we received from the Columbus Police Department, which are of much poorer quality (and representative of actual latents) than ones that we had produced ourselves. Against the high quality (clean and relatively deformation free) latent that we had produced the algorithm selects the true matches easily.

Figure 10.15: Sample latent print LCP10.

Table 10.1: Matching results: file print MAQ1, latent LCP10.

File print	MAQ1			DBB1			KAL7		
Status	True match			False match			False match		
Score Basepoint	Total	Min	TCWR	Total	Min	TCWR	Total	Min	TCWR
[225, 244]	400	20	20	0			0		
[245, 265]	410	20	20	5.3	1	5.3	0		
[317, 328]	29	2	14	0			0		
[316, 359]	96	5	19	0			0		
[282, 394]	0			0			0		
[303, 285]	180	10	18	0			19	6	3.1
Average	190			0.88			3.2		

Figure 10.16: Sample file print MAQ1.

Table 10.2: Matching results: file print MAP6, latent LCP13.

File print	MAP6CLN			MAK3			KAC7		
Status	True match			False match			False match		
Score Basepoint	Total	Min	TCWR	Total	Min	TCWR	Total	Min	TCWR
[206, 346]	110	21	5.3	37	4	9.4	0		
[285, 364]	160	23	6.8	0			0		
[308, 295]	99	17	5.8	21	14	1.5	71	8	8.9
[237, 293]	42	8	5.3	55	6	9.1	0		
[237, 266]	130	23	5.8	87	10	8.7	2	1	2.0
[288, 273]	130	23	5.8	89	10	8.9	100	12	8.7
Average	110			48			30		

Table 10.3: Matching results: file print MAQ1, latent LCP14.

File print	MAQ1			DAZ1		
Status	True match			False match		
Score Basepoint	Total	Min	TCWR	Total	Min	TCWR
[240, 335]	68	7	9.7	0		
[202, 368]	110	11	9.9	36	6	6.1
[239, 421]	120	12	9.8	0		
[241, 365]	0			0		
[266, 323]	0			0		
[255, 383]	110	10	11	36	6	6.0
Average	67			12		

Table 10.4: Matching results: file print MAQ6, latent LCP9.

File print	MAQ6			DBE9		
Status	True match			False match		
Score Basepoint	Total	Min	TCWR	Total	Min	TCWR
[224, 223]	3	3	0.97	27	13	2.1
[297, 355]	120	7	17	5.8	2	2.9
[177, 329]	120	7	17	0		
[309, 279]	150	9	16	0		
[162, 281]	190	11	17	0		
[256, 302]	0			0		
Average	96			9.5		

Table 10.5: Matching results: file print DAB1, latent LCP8.

File print	dab1			dax3			daa1		
Status	True match			False match			False match		
Score Basepoint	Total	Min	TCWR	Total	Min	TCWR	Total	Min	TCWR
[216, 307]	0			0			0		
[245, 355]	0			0			23	9	2.7
[202, 381]	0			0			0		
[257, 395]	0			16	4	4.1	0		
[297, 355]	0			0			0		
[288, 311]	8.0	6	1.3	50	13	3.9	10	3	3.2
Average	1.3			11.0			9.7		

CHAPTER XI

Summary to Part II

Part II of this dissertation deals with feature extraction and identification on a variety of images, including examples on radiographs and membrane images from a scanning electron microscope. In particular, however, this part studies in detail defect detection on printed circuit boards and latent fingerprint identification.

A first step in this work is a study of preprocessing methods for image enhancement and topological detail extraction from digital images. The culmination of our work in this area was an original method for the extraction of flow and curvature information by applying the methods of least squares minimization to level curves of the image. The usefulness of this technique was shown via its use for orientation specific filtering and edge detection. In fact, this work has been accepted for publication [70].

With regards to feature extraction and identification, we first considered defect detection on printed circuit boards. The presentation in this dissertation was culled from some work this author did with Alan Sprague [1]. This work devel-

oped a reference comparison approach to defect detection that differs from existing algorithms in that it does not rely on computational expensive morphological transformations (e.g., edge erosion and/or dilation). This algorithm was implemented on an IBM PC based platform by Prof. Sprague and this author, and results of that implementation are included here.

This dissertation also presents new ideas in latent fingerprint identification. Existing commercial systems are expensive, proprietary, and in general do not perform satisfactorily on latent fingerprints. Our approach uses the flow and curvature information available from our previous work to generate a sophisticated feature segment structure from which fingerprint minutiae are extracted. Of special interest here is an original classification system of structure defects in fingerprint ridges. The minutiae extraction results compare favorably with those from commercial products. Next a complex feature matching algorithm is developed that matches not only fingerprint minutiae, which are sensitive to morphological changes in the finger caused by skin elasticity, but also performs a complex evaluation/comparison of the flow, curvature, and ridge width surfaces of the latent and comparison fingerprint. This provides an accurate identification method which is illustrated by results on a library of 700 prints.

CHAPTER XII

Summary and future work

This dissertation studies and presents new results in computer aided tomography and feature detection and recognition. These are areas of increasing importance in nondestructive evaluation. Part I deals with radiographic and tomographic images. Included here are an original radiograph simulation technique and several new developments in limited angle tomography. In particular a method of analytic continuation, suggested in a short mathematical paper by Palamdov and Denisjuk [4], is implemented. To the best of our knowledge this is the first actual realization of this approach, which required some novel solutions to overcome numerical instabilities.

Also studied is the use of a priori information in limited angle reconstructions. Due to inherent inflexibilities in Fourier transform methods (including the analytic continuation method just mentioned), a priori information is handled with an iterative (also known as algebraic) reconstruction method. To allow rapid convergence and still guarantee convergence with experimental data we introduce to the itera-

tion a relaxation parameter which decays geometrically to zero. We also provide quantitative results on experimental data showing the importance of proper ordering of the projection data and selection of the initial iterate. Also a new result is given showing how proper use of density restrictions (i.e., disallowing densities less than 0 or bigger than some predetermined maximum) can prevent divergence on long iterate reconstructions.

A natural extension of the tomography work would be methods for automatic detection of flaws (or other features) in radiographs and tomographic reconstructions. Our research funding, however, was toward different but parallel problems: automated feature recognition and identification in planar images, chiefly printed circuit boards and fingerprints. This is the material comprising Part II. As first step in this area we study preprocessing methods for image enhancement and topological detail extraction from digital images. The culmination of our work in this area was an original method for the extraction of flow and curvature information by applying the methods of least squares minimization to level curves of the image. The usefulness of this technique was shown via its use for orientation specific filtering and edge detection, including an examples on a radiographic images. This work has been accepted for publication [70].

Identification via feature based methods has applications in many areas. We consider feature extraction and identification on images of printed circuit boards (for defect detection) [1] and images of fingerprints (for latent fingerprint iden-

tification). The work on printed circuit board images features the development of an original reference comparison technique for defect detection that does not require computationally expensive morphological transformations (e.g., edge erosion and/or dilation). Fingerprint images, though superficially similar to images of printed circuit boards, present many new problems. Feature extraction is complicated by the presence of structural noise in fingerprint ridges, while latent identification is hampered by morphological changes resulting from skin elasticity. To overcome these problems an original ridge structural defect classification method and a complex comparison scheme are developed, the latter making extensive use of flow and curvature information extracted using the techniques developed in the first chapter of Part II.

There are several areas of future work which can be explored. The numerical stability of the analytic continuation method of Section 4.1 could possibly be improved by a reformulation of the Cauchy integral from which it stems or perhaps some other theoretical considerations. The quantitative iterative reconstruction results should be checked against more experimental data, and the use of density restrictions to control divergence should have some theoretical underpinnings. The fingerprint extraction algorithm could use the defect classification method to not only identify defects but to also correct them. In this matter the procedure could be made iterative, with each iteration correcting more defects until only true features remain. For this purpose and also for latent feature comparisons a de-

tailed study of the effects of pressure and skin elasticity on fingerprint registration would be invaluable. Moreover, the feature extraction and identification methods presented here may be adapted for the purpose of automated defect detection on radiographs and tomographs.

Appendix A

Radiograph simulation program details

A.1 Base element attenuation subroutines

A wide variety of objects can be constructed from the 6 base types and cut-planes, especially when one realizes that in addition to specifying the orientation and location of each element, one also specifies the density of the object. One may specify a negative density, which effectively performs object subtraction. For example, a hollow pipe can be formed from two concentric cylinders, the inner cylinder having density equal to the negative of the first.

Even though it is possible to simulate pipes with the current base elements, if one were doing many simulations with pipes, then it would be useful to have a base element of type `PIPE`. This would simplify object positioning and decrease simulation calculation times. The simulation package is structured to allow for easy addition of new element types. As an introduction to this topic, and as an aid to understanding the base element types themselves, let us examine the implementation of an existing element type, the ellipsoid.

As detailed in the section on 2D simulations, we do not code the attenuation for base elements in arbitrary orientations. We code, rather, for elements in a canonical position and account for arbitrary orientations by adjusting the coordinates of the intersecting ray. The canonical position for the ellipsoid is chosen to be centered at the origin, with ellipsoid axes coinciding with the coordinate axes, as illustrated in Fig. A.1. This ellipsoid has half-axis lengths of e_1, e_2, and e_3, so the defining equation for the ellipsoid is

$$(x_1/e_1)^2 + (x_2/e_2)^2 + (x_3/e_3)^2 \leq 1. \tag{A.1}$$

The equation for a line $L_{\vec{v},\vec{w}}$ in 3-space is the same as that given in Eq. 2.6, except now $\vec{v}$ and $\vec{w}$ are 3-dimensional vectors. Inserting this parameterization into Eq. A.1 and solving yields the line-ellipsoid intersection points

$$s_{\min} = \frac{-b - \Delta}{a} \qquad s_{\max} = \frac{-b + \Delta}{a}, \tag{A.2}$$

where

$$\begin{aligned} a &= \sum_{i=1}^{3} v_i^2/e_i^2 \\ b &= \sum_{i=1}^{3} v_i w_i/e_i^2 \\ c &= \left(\sum_{i=1}^{3} w_i^2/e_i^2\right) - 1 \\ \Delta &= \sqrt{b^2 - ac}. \end{aligned}$$

The code implementing this calculation is shown in Fig. A.2. The imports are two structure pointers. The first (pointer `ell`), points to a structure containing the

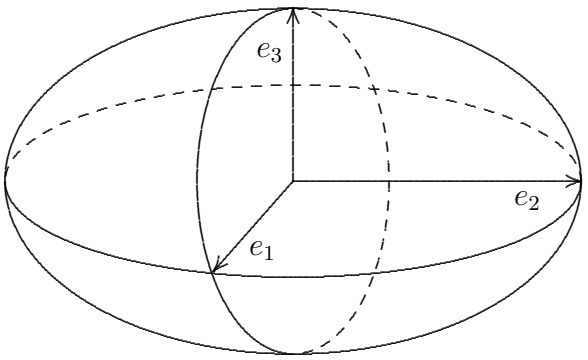


Figure A.1: Illustration of the ellipsoid $(x_1/e_1)^2 + (x_2/e_2)^2 + (x_3/e_3)^2 \leq 1$.

defining information for this instance of the ellipsoid element (size, orientation, position, density), and the second (pointer `ray`) defines the (potentially intersecting) line.

The first block of code is a crude check to see if the line `*ray` is even close to the ellipsoid. If the ellipsoid is small, then many of the import lines will completely miss the ellipsoid. This check provides a quick exit from this routine for those lines.

The next two blocks compute the values of $s_{\min}$ and $s_{\max}$ (`t1` and `t2` respectively), which are sent to the routine `plane_limits()` for modifications due to any cut-plane restrictions. (The routine `plane_limits` is a general purpose routine used by all the element attenuation routines that incorporate cut-planes.) Finally, the total length is multiplied by the element density and the result (the linear attenuation) is returned.

```
/************************ ELLIPSOID ***************************/
float ellipsoid(ELEMENT *ell,LINE *ray)
   /* This routine returns the length of the intersection       */
   /* between the ellipsoid ell and the line *ray.  The object  */
   /* ell definition includes the size, density and orientation */
   /* of the ellipsoid.                                         */
   {
   int i;
   float eff_length,a,b,c,det,t1,t2;
   LINE newray;

   /* Crude out-of-bounds check */
   for(i=0;i<3;i++)
      if(fabs(ray->offset[i])<=ell->param[i]) break;
   if(i>2) return 0.0;

   /* Convert to elliptical coordinates */
   for(i=0;i<3;i++)
      {
      newray.dir[i]=ray->dir[i]/ell->param[i];
      newray.offset[i]=ray->offset[i]/ell->param[i];
      }

   a=dot(newray.dir,newray.dir);
   b=dot(newray.dir,newray.offset);
   c=dot(newray.offset,newray.offset)-1.0;

   /* Compute crossing times */
   det=b*b-a*c;
   if(det<TOO_SMALL*TOO_SMALL) return 0.0;   /* No intersection! */
   det=sqrt(det);
   t1=(-b-det)/a; t2=(-b+det)/a;

   /* Incorporate "cut plane" restrictions */
   plane_limits(&t1,&t2,ell,ray);

   /* Compute total length, including density */
   if(t2<t1) eff_length=0;
    else     eff_length=(t2-t1)*ell->density;

   return eff_length;
   }
```

Figure A.2: Subroutine for calculation of ellipsoid linear attenuation.

A.2 Program organization

We now explain the program organization. Follow this discussion along with Fig. A.3, the program flowchart.

At the start of program execution, the user is prompted for X-ray source type, location, orientation, and the name of the data file containing the element descriptions. Next the program reads the data file and stores the elements in an array of structures of type `ELEMENT`.

After obtaining the input parameters, the program loops through the list of elements. For each element, the program shifts the source position relative to the object's orientation and position. (This is faster than generating each ray and shifting each ray independently.) The ray generation routine then calculates the parameterization vectors $\vec{v}$ and $\vec{w}$ (refer to Eq.2.6) for each ray incident on the display region, and sends this parameterization information to the proper element attenuation routine. The return value from the attenuation routine (the linear attenuation) is summed into the corresponding position in the display array `proj[ ][ ]`. After this has been done for every ray in the display region, the program loops back to repeat the process for the next element.

Once the linear attenuations for each element have been calculated and summed into the display array `proj[ ][ ]`, the exponential of the attenuations is calculated. (Recall that the physical attenuation of an X ray is proportional to the

exponential of the linear attenuation.) The display routine is then called to output the simulation on the view screen.

A.3 Data structures

The data structures used by this program are defined in the header file `DXT.H`. In this section we give a description of the most important ones.

`VECTOR` This structure is an array of 3 elements of type `float`. It represents a position independent vector in 3-space.

`POINT` This structure is also an array of 3 elements of type `float`. This one represents a point in 3-space. A `POINT` is translation dependent, whereas a `VECTOR` is translation independent.

`LINE` This structure consists of two `VECTOR`'s which correspond to the vector parameterization of a line. The first is a unit vector parallel to the direction of the line, and the second is the offset of the line from the origin. These two vectors should be orthogonal. (Refer to Eq. 2.6).

`TRANSFORM` This structure is a 3×3 array of type `float`, corresponding to a linear transformation on three space (i.e., an element of the general linear group GL^3).

`ORIENTATION` This structure represents rigid body motions in 3-space. It consists of a `TRANSFORM` and a `VECTOR`. The `TRANSFORM` should be an element of SL^3

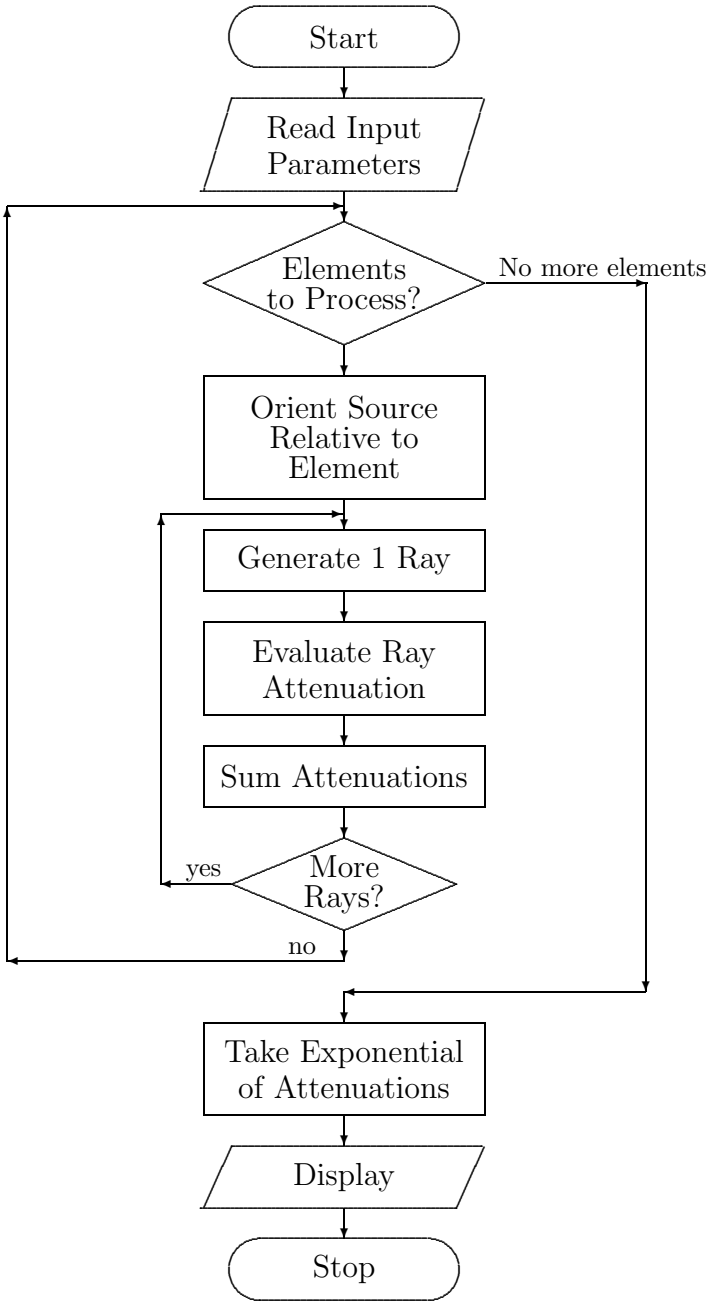


Figure A.3: Flowchart for radiograph simulation package

(an orthogonal matrix with determinant +1) while the `VECTOR` can be any 3 dimensional vector. This structure is used to define the orientation and position of the base elements relative to their canonical position. (Refer to the section on base element types.) The rigid body motion is performed by starting with the element in its canonical orientation, rotate it according to the `TRANSFORM` and **THEN** translate by the offset in `VECTOR`. Note that these operations are not commutative, so the order of operations is important.

`CUT_PLANE` This structure contains the information necessary to determine cut-plane restrictions on an element. It consists of a `VECTOR` $\vec{n}$ that is normal to the cut-plane, the offset of the plane from the origin (of type `float`), and an integer value, $\sigma = \pm 1$, which specifies the half space to exclude. The half space containing the point $(\text{offset} + \sigma) \times \vec{n}$ is excluded.

`ELEMENT` This structure contains all the information necessary to specify a base element. It consists of an `ORIENTATION` specifying the orientation and position of the element from its canonical position, the density of the element, an array of up to (currently) 5 element specific parameters of type `float`, an integer count of the number of cut-planes, an array of up to (currently) 10 `CUT_PLANES`, and a pointer to the linear attenuation evaluation function for the element (which is determined by the input routine from the element type descriptor in the object description data file).

For specifics refer to the file `DXT.H` and the documentation and program listings for specific base elements.

A.4 Attenuation subroutine format

Element linear attenuation evaluation functions are stored in the file `ELEMENTS.MSC`. Each function has two imports. The first import is a pointer to an `ELEMENT` variable describing the base element for which the attenuation is to be calculated, and the second is a pointer to a `LINE` variable defining the path of the X ray for which the linear attenuation is required. The `LINE` is assumed to be already transformed to the local coordinates for the element (i.e., the element can be assumed to be in its canonical position), so only the density and element specific parameters need to be accessed. The return value of the attenuation subroutine is the linear attenuation of the X ray. It is the responsibility of the attenuation subroutine to include cut-plane restrictions if the element type supports them. Cut-plane restriction can be calculated by sending the X ray element entry and exit times (relative to the import line parameterization) to the subroutine `PLANE_LIMITS`. If the element type does not support cut-planes then the attenuation function does not need to explicitly calculate the X ray entry and exit times.

A.5 Ray generation subroutine format

The ray generation subroutines are contained in the file `RAYGEN.MSC`. These routines generate the parameters of the lines corresponding to X-ray paths in the display region and send this information to the base element linear attenuation functions.

The imports to subroutines of this class are the source orientation/position, a base element descriptor (`ELEMENT` data type), and the display array `proj[ ][ ]` that stores the sum of the linear attenuations for each displayed pixel. (It is the responsibility of the calling routine to initialize this array to zero before the first call to these functions.) Each call to these subroutines causes the linear attenuation due to one element to be summed into the display array.

The X rays are generated according to the following scheme. Take the line through the origin coincident with the x-axis, and transform it according to the source orientation. The plane perpendicular to this line that passes through the origin (in the original coordinate system) is the display plane. The display region is a rectangle on this plane, which is centered about the intersection of this plane with the transformed line. Now generate one X ray for each pixel in the display region according to the source beam geometry. For example, the parallel beam generation routine generates an X ray through each display pixel that is parallel to the center line defined above. On the other hand, the cone beam generation

routine generates for each display pixel an X ray that passes through both the display pixel and the source location.

Each generated X ray must be sent to the element linear attenuation function. Recall that these functions require that the input X ray be transformed to the element local coordinate system. One could generate the X rays as outlined in the preceding paragraph, and then transform each according to the element orientation, but this is time consuming. A faster way (and the way the supplied ray generation subroutines work) is to transform the source and display region to the element local coordinate system, and then generate the X rays directly in the local coordinate system.

After each (transformed) X ray is generated it is sent to the proper element linear attenuation function, and the attenuation is summed into the display attenuation array `proj[ ][ ]`. When this is completed for the entire display region for this one element, the ray generation routine passes control back to the calling routine.

Bibliography

[1] A. P. Sprague, M. J. Donahue, and S. I. Rokhlin, "A method for automatic inspection of printed circuit boards," *Computer Vision, Graphics and Image Processing: Image Understanding*, **54**, 401–415 (1991).

[2] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988.

[3] L. Feldkamp, L. Davis, and J. Kress, "Practical cone-beam algorithm," *Journal of the Optical Society of America*, **1**, 612–619 (1984).

[4] V. Palamodov and A. Denisjuk, "Inversion de la transformation de Radon d'après des données incomplètes," *C. R. Acad. Sci. Paris*, **307**, 181–183 (1988).

[5] J. Radon, "Über die Bestingmmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten," *Ber. Verh. Sächs. Akad. Wiss. Leipsiz, Math-Nat. kl.*, **69**, 262–277 (1917).

[6] R. N. Bracewell and A. C. Riddle, "Strip integration in radio astronomy," *Aus. J. Phys.*, **9**, 198–217 (1956).

[7] R. N. Bracewell and A. C. Riddle, "Inversion of fan-beam scans in radio astronomy," *The Astrophysical Journal*, **150**, 427–434 (1967).

[8] A. Cormack, "Representation of a function by its line integrals, with some radiological applications," *J. Appl. Phys.*, **34**, 2722–2727 (1963).

[9] A. Cormack, "Representation of a function by its line integrals, with some radiographical applications II," *J. Appl. Phys.*, **35**, 195–207 (1964).

[10] G. N. Hounsfield, "Computerized transverse axial scanning tomography: Part I, description of the system," *Br. J. Radiol.*, **46**, 1016–1022 (1973).

[11] G. N. Ramachandran and A. V. Lakshminarayanan, "Three-dimensional reconstruction from radiographs and electron micrographs: application of convolutions instead of Fourier transforms," *Proc. Nat. Acad. Sci. US*, **68**, 2236–2240 (1971).

[12] L. A. Shepp and B. F. Logan, "The Fourier reconstruction of a head section," *IEEE Trans. Nucl. Sci.*, **NS-21**, 21–43 (1974).

[13] S. R. Deans, *The Radon Transform and Some of Its Applications.* Wiley, 1983.

[14] F. Natterer, *The Mathematics of Computerized Tomography.* John Wiley & Sons Ltd, 1986.

[15] R. M. Lewitt, R. H. T. Bates, and T. M. Peters, "Image reconstruction from projections: III: projection completion methods (theory)," *Optik*, **50**, 180–205 (1978).

[16] A. K. Louis, "Ghosts in tomography—the null space of the Radon transform," *Math. Meth. in the Appl. Sci.*, **3**, 1–10 (1981).

[17] A. K. Louis, "Nonuniqueness in inverse Radon problems: the frequency distribution of the ghosts," *Math. Z.*, **185**, 429–440 (1984).

[18] H. K. Tuy, "Reconstruction of a three-dimensional object from a limited range of views," *J. Math. Anal. Appl.*, **80**, 598–616 (1981).

[19] K. T. Smith, "Inversion of the x-ray transform," *SIAM-AMS Proceedings*, **14**, 41–52 (1984).

[20] M. E. Davison and F. A. Grünbaum, "Tomographic reconstructions with arbitrary directions," *Comm. Pure Appl. Math.*, **34**, 77–120 (1981).

[21] M. E. Davison, "The ill-conditioned nature of the limited angle tomography problem," *SIAM J. Appl. Math.*, **43**, 428–448 (1983).

[22] L. E. Bryant and P. McIntire, eds., *Radiography and Radiation Testing.* Vol. 3, American Society for Nondestructive Testing, 1985.

[23] S. Kaczmarz, "Angenaherte Auflosung von Systemen Linearer Gleichungen," *Bull. Acad. Pol. Sci. Lett. A*, **6-8A**, 355–357 (1937).

[24] A. S. Glassner, ed., *An Introduction to Ray Tracing.* Academic Press, 1989.

[25] L. Feldkamp and G. Jension, "3-d x-ray computed tomography," *Review of Progress in Quantitative Nondestructive Evaluation (QNDE)*, **5A**, 555–566 (1986).

[26] L. Wang, *Three Dimensional Computer Aided Tomography of Welds.* Master's thesis, The Ohio State University, 1988.

[27] R. Gordon and G. T. Herman, "Three-dimensional reconstruction from projections: a review of algorithms," *International Review of Cytology*, **38**, 111–151 (1974).

[28] M. Chiu, H. H. Barrett, and R. G. Simpson, "Three-dimensional reconstruction from planar projections," *J. Opt. Soc. Am.*, **70**, 755–762 (1980).

[29] B. D. Smith, "Image reconstruction from cone-beam projections: necessary and sufficient conditions and reconstruction methods," *IEEE Trans. Med. Imaging*, **MI-4**, 14–25 (1985).

[30] H. K. Tuy, "An inversion formula for cone-beam reconstruction," *SIAM J. Appl. Math.*, **43**, 546–552 (1983).

[31] D. V. Finch, "Cone beam reconstruction with sources on a curve," *SIAM J. Appl. Math.*, **45**, 665–673 (1985).

[32] G. DiChiro, R. A. Brooks, L. Dubal, and E. Chew, "The apical artifact: elevated attenuation values toward the apex of the skull," *J. Comput. Assist. Tomog.*, **2**, 65–79 (1978).

[33] W. D. McDavid, R. G. Waggener, W. H. Payne, and M. J. Dennis, "Correction for spectral artifacts in cross-sectional reconstruction from x-rays," *Med. Phys.*, **4**, 54–57 (1977).

[34] A. J. Duerinckx and A. Macovski, "Polychromatic streak artifacts in computed tomographic images," *J. Comput. Assist. Tomog.*, **2**, 481–487 (1978).

[35] W. Rudin, *Real and Complex Analysis*. McGraw-Hill, 2nd ed., 1974.

[36] M. J. Donahue, *The Angle Between the Null Spaces of the Radon and Related Transforms*. PhD thesis, The Ohio State University, 1991.

[37] A. Browne and L. Norton-Wayne, *Vision and Information Processing for Automation*. Plenum, 1986.

[38] R. T. Chin, "Survey: automated visual inspection: 1981–1987," *Computer Vision, Graphics, and Image Processing*, **41**, 346–381 (1988).

[39] T. Pavlidis, "A review of algorithms for shape analysis," *Computer Graphics and Image Processing*, **7**, 243–258 (1978).

[40] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. Academic Press, 1982.

[41] J. Sanz and A. Jain, "Machine vision techniques for inspection of printed wiring boards and thick-film circuits," *J. Opt. Soc. Am. A*, **3**, 1465–1482 (1986).

[42] G. A. W. West, "A system for the automatic visual inspection of bare-printed circuit boards," *IEEE Trans. Systems, Man, and Cybernetics*, **SMC-14**, 767–773 (1984).

[43] J. R. Mandeville, "Novel method for analysis of printed circuit images," *IBM J. Research and Development*, **29**, 73–86 (1985).

[44] Q. Z. Ye and P. E. Danielsson, "Inspection of printed circuit boards by connectivity preserving shrinking," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **PAMI-10**, 737–742 (1988).

[45] A. M. Darwish and A. K. Jain, "Rule based approach for visual pattern inspection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **PAMI-10**, 56–68 (1988).

[46] Y. Hara, N. Akiyama, and K. Karasaki, "Automatic inspection system for printed circuit boards," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **PAMI-5**, 623–630 (1983).

[47] H. Yoda, Y. Ohuchi, Y. Taniguchi, and M. Ejiri, "An automatic wafer inspection system using pipelined image processing techniques," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **PAMI-10**, 4–16 (1988).

[48] L. F. Pau, "Integrated testing and algorithms for visual inspection of integrated circuits," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **PAMI-5**, 602–608 (1983).

[49] U.S. Dept. of Justice, Federal Bureau of Investigation, *The Science of Fingerprints: Classification and Uses*, ch. 2, 5–79. U.S. Government Printing Office, 1984.

[50] S. P. Morse, "A mathematical model for the analysis of contour-line data," *Journal of the Association for Computing Machinery*, **15**, 205–220 (1968).

[51] T. K. Peucker and D. H. Douglas, "Detection of surface-specific points by local parallel processing of discrete terrain elevation data," *Computer Graphics and Image Processing*, **4**, 375–387 (1975).

[52] M. Kawagoe and A. Tojo, "Fingerprint pattern classification," *Pattern Recognition*, **17**, 295–303 (1984).

[53] A. S. Rabinowitz, "Fingerprint card search results with ridge-contour based classification," in *5th International Conference on Pattern Recognition*, 475–477, IEEE, 1980.

[54] E. Peli, "Adaptive enhancement based on a visual model," *Optical Engineering*, **26**, 655–660 (1987).

[55] M. Hueckel, "An operator which locates edges in digital pictures," *Journal of the Association for Computing Machinery*, **18**, 113–125 (1971).

[56] F. O'Gorman, "Edge detection using walsh functions," *Artificial Intelligence*, **10**, 215–223 (1978).

[57] A. Rosenfeld and E. Johnston, "Angle detection on digital curves," *IEEE Transactions on Computers*, **22**, 875–878 (1973).

[58] P. Parent and S. W. Zucker, "Trace inference, curvature consistency, and curve detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**, 823–839 (1989).

[59] S. W. Zucker, "Early orientation selection: tangent fields and the dimensionality of their support," *Computer Vision, Graphics, and Image Processing*, **32**, 74–103 (1985).

[60] D. G. Luenberger, *Optimization by Vector Space Methods*. Wiley, 1969.

[61] R. M. Haralick and L. Watson, "A facet model for image data," *Computer Graphics and Image Processing*, **15**, 113–129 (1981).

[62] L. J. Kitchen and J. A. Malin, "The effect of spatial discretization on the magnitude and directional response of simple differential edge operators on a step edge," *Computer Vision, Graphics and Image Processing*, **47**, 243–258 (1989).

[63] K. V. Mardia, *Statistics of Directional Data*. Academic Press, 1972.

[64] R. Jain, "Direct computation of the focus of expansion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **5**, 58–64 (1983).

[65] K. Prazdny, "On the information in optical flows," *Computer Vision, Graphics, and Image Processing*, **22**, 239–259 (1983).

[66] S. Negahdaripour and B. K. P. Horn, "A direct method for locating the focus of expansion," *Computer Vision, Graphics, and Image Processing*, **46**, 303–326 (1989).

[67] W. Burger and B. Bhanu, "Estimating 3-d egomotion from perspective image sequences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, 1040–1058 (1990).

[68] P. Y. Simard and G. E. Mailloux, "Vector field restoration by the method of convex projections," *Computer Graphics and Image Processing*, **52**, 360–385 (1990).

[69] F. Harary, *Graph Theory*. Addison-Wesley, 1969.

[70] M. J. Donahue and S. I. Rokhlin, "On the use of level curves in image analysis," 1992. To appear in CVGIP.